a = [1, 2, 3]
iterator1 = iter(a)
next (iterator1)
1
a.remove(2)
a
>[1, 3]
next(iterator1)
→ $\boxed{\text{s fup iteration}}$

a = [1, 2, 3]
b = [10, 20, 30]
iterator1 = iter(a)
next (iterator1)
>1
a = b
next (iterator 1)
$\boxed{2}$

for <> in <iterable>
<iterator>

Iterators, Generators, and Streams

Computer Science Mentors CS 61A

April 22 to April 24, 2019

1 Iterators and Generators

Gen func $\xrightarrow{()}$ Gen obj
(type of iterator)
yield
yield from <gen obj>
<iterator>
<iterable>

Iterable: object you can go over item-by-item

iterable $\xrightarrow{iter(\cdot)}$ iterator

container allows next ( )

Ex: a = range(3)
iterator1 = iter(a)
iterator 2 = iter(a)
< these are separate iterators; draw them out >

Ex: an_iterable = [1,2,3]
iterator1 = iter(an_iterable)
next (iterator1)
an_iterable[1] = 20

p
skip: 3

1. What Would Python Display?

```
class SkipMachine:
    skip = 1
    def __init__(self, n=2):
        self.skip = n + SkipMachine.skip

    def generate(self):
        current = SkipMachine.skip
        while True:
            yield current
            current += self.skip
            SkipMachine.skip += 1

p = SkipMachine()
twos = p.generate()
SkipMachine.skip += 1
twos2 = p.generate()
threes = SkipMachine(3).generate()
```

~~1~~ ~~2~~ ~~3~~ ~~4~~ 5

p = SkipMachine() → <SM instance>
twos = p.generate() → <GO>
twos2 = p.generate() → <GO>
doesn't change P.Skip
SkipMachine(3): skip = 5

(a) **next**(twos)
2

(b) **next**(threes)
2

(c) **next**(twos)
2 + p.skip = 5

next(iterator1)
$\boxed{20}$

(d) **next**(twos)

5 + p.skip = 8

(e) **next**(threes)

2 + SkipMachine(3).skip = 7

(f) **next**(twos2)

5

2. What does the following code block output?

```
def foo():
    a = 0
    if a < 10:
        print("Hello")
        yield a
        print("World")

for i in foo():
    print(i)
```

$\rightarrow$ GO (iterator) [0]

Hello
0
World

```
> f = foo()
> list(f)
Hello
World
> [0]
```

```
> f = foo()
> next(f)
Hello
0
> next(f)
World
StopIteration Error.
```

3. How can we modify `foo` so that it satisfies the following doctests?

```
>>> a = list(foo())
>>> a
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
def foo():
    for a in range(1, 11):
        yield a
```

4. Define `filter_gen`, a generator that takes in iterable `s` and one-argument function `f` and yields every value from `s` for which `f` returns `True`

```
def filter_gen(s, f):
    """
    >>> list(filter_gen([1, 2, 3, 4, 5],
                             lambda x: x % 2 == 0))
    [2, 4]
    >>> list(filter_gen((1, 2, 3, 4, 5), lambda x: x < 3))
    [1, 2]
    """
    for x in s:
        if f(x):
            yield x
```

2 Streams

1. (a) What are the advantages or disadvantages of using a stream over a linked list?

   lazy

   (b) What's the maximum size of a stream?

   (c) What's stored in `first` and `rest`? What are their types?

   val another stream

   (d) When is the next element actually calculated?

   when requested.

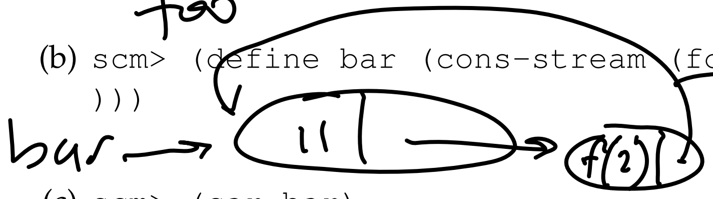
2. What Would Scheme Display?

(a) `scm> (define (foo x)(+ x 10))`

foo

(b) `scm> (define bar (cons-stream (foo 1)(cons-stream (foo 2)bar )))`

11 12



(c) `scm> (car bar)`

11

(d) `scm> (cdr bar)`

promise

(e) `scm> (define (foo x)(+ x 1))`

foo

(f) `scm> (cdr-stream bar)`

f(2) (3 . promise(uf))

(g) `scm> (define (foo x)(+ x 5))`

foo

☆ (h) `scm> (car bar)`

11 (already calculated in (c))

☆ (i) `scm> (cdr-stream bar)`

(3 . promise (uf)) (lazily evaluated)

(j) `scm> (cdr bar)`

promise (fixed)

3 Code Writing for Streams

1. Implement `double-naturals`, which is a returns a stream that evaluates to the sequence 1, 1, 2, 2, 3, 3, etc.

Write out cons structure

```
(define (double-naturals)
    (double-naturals-helper 1 #f)
)
(define (double-naturals-helper first go-next)
    (if go-next
        (cons-stream first (dnh (+ first 1) #f))
        (cons-stream first (dnh first #t)
    )
)
```

2. Implement `interleave`, which returns a stream that alternates between the values in stream1 and stream2. Assume that the streams are infinitely long.

```
(define (interleave stream1 stream2)
    (cons-stream
        (car stream1)
        (interleave stream2
            (cdr-stream stream1))
    )
)
```