TRUSTYOU

Things I wish I Knew Before Using Python for Data Processing

Miguel Cabrera
@mfcabrera

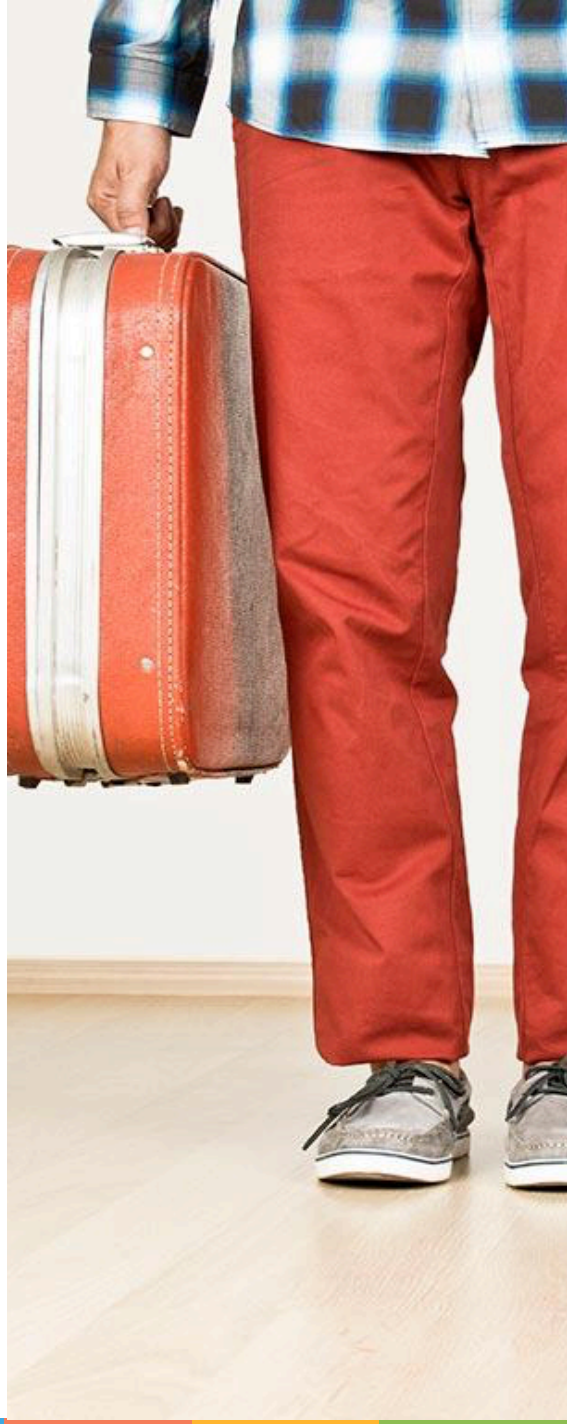
20 July 2016

Hello!

I am Miguel!

Data Engineer / Scientist @ TrustYou
Python ~ 2 years
Berlin

@mfcabrera
mfcabrera at gmail
http://mfcabrera.com



Priors!

- (Relatively) New to Python, mostly Scientific stack
- You have used things like Numpy, Scikit-Learn, Gensim, etc…
- Your job title includes either the word Data or “Machine Learning”.
- Not necessarily a trained Software Engineer

Who is who?

- Data Scientist?
- Data Analyst?
- Data Engineer?
- Machine Learning Developer?
- Software Developer?
- Other?

Agenda

- Basic Concepts and practices
- Some goodies of the collection module
- Iterators and Iterables
- Conclusion

David's Story

- Recent university grad
- Mostly R and Matlab
- Writes nifty code to classify documents using Jupyter Notebooks
- Mostly NltK and Scikit-Learn

localhost:8888/notebooks/interact_basic.ipynb#

radicalcartography News Blogs Accounts Continuum Facebook SA ucp contouring draw.io Bokeh bokeh/bokeh

jupyter interact_basic Last Checkpoint: 01/08/2015 (autosaved)

File Edit View Insert Cell Kernel Help Python 3

Code Cell Toolbar: None

In [1]:
```
from bokeh.models import ColumnDataSource
from bokeh.plotting import *

import numpy as np
```

In [2]:
```
x = np.linspace(0, 2*np.pi, 2000)
y = np.sin(x)
```

In [3]:
```
output_notebook()
```

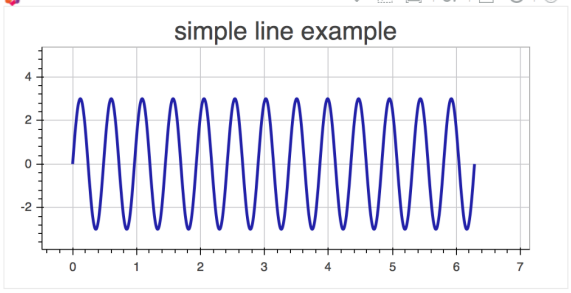
BokehJS successfully loaded.

In [4]:
```
source = ColumnDataSource(data=dict(x=x, y=y))

p = figure(title="simple line example", plot_height=300, plot_width=600)
p.line(x, y, color="#2222aa", line_width=3, source=source, name="foo")
```

Out[4]: <bokeh.plotting.Figure at 0x10a033860>

In [5]:
```
def update(f, w=1, A=1, phi=0):
    if   f == "sin": func = np.sin
    elif f == "cos": func = np.cos
    elif f == "tan": func = np.tan
    source.data['y'] = A * func(w * x + phi)
    source.push_notebook()
```

In [6]:
```
show(p)
```



In [7]:
```
from IPython.html.widgets import interact
interact(update, f=["sin", "cos", "tan"], w=(0,100), A=(1,10), phi=(0, 10, 0.1))
```

f sin

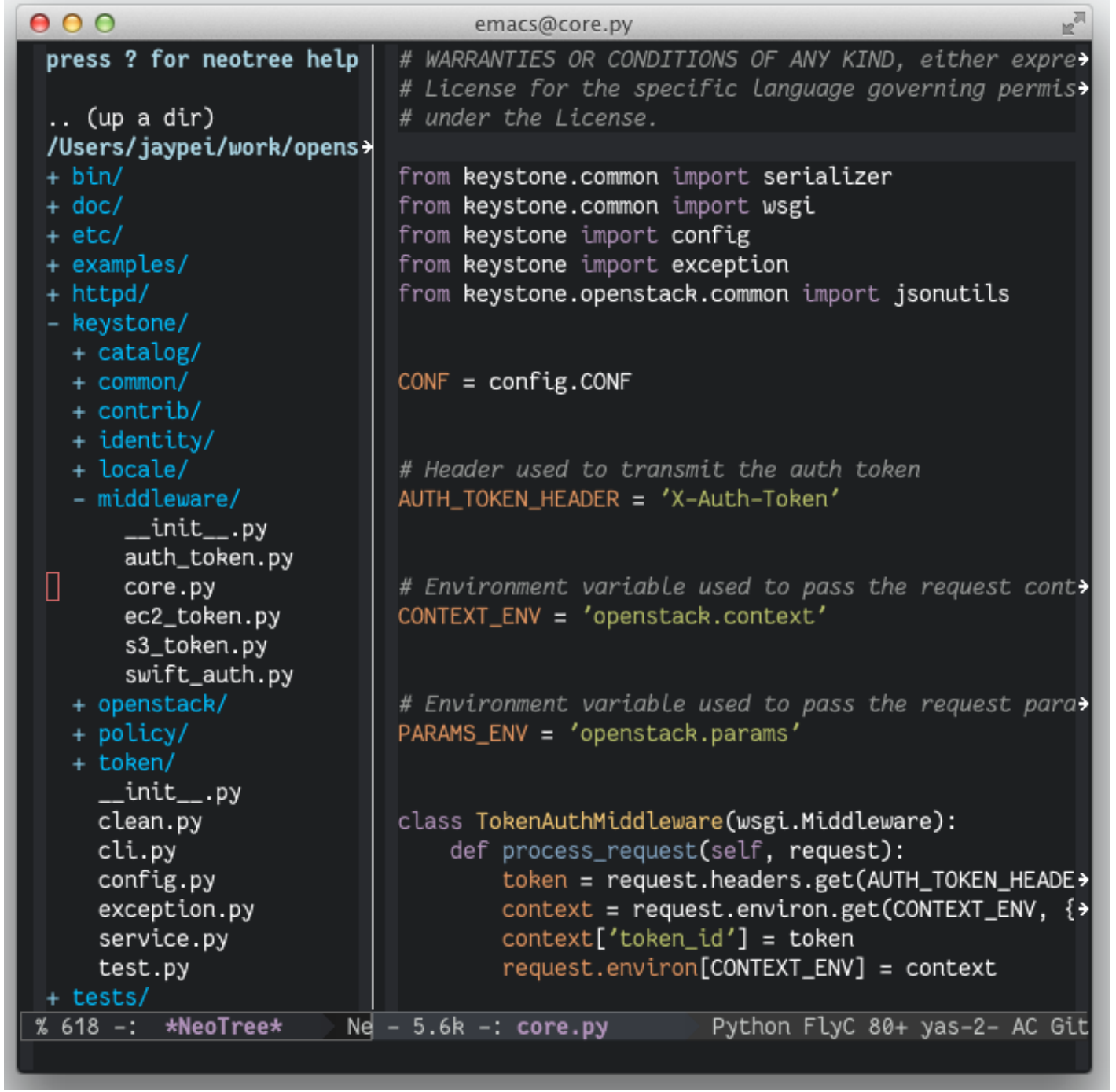
w 13

A 3

phi 0

In [ ]:

Open # on this page in a new tab


emacs@core.py
press ? for neotree help
.. (up a dir)
/Users/jaypei/work/opens
+ bin/
+ doc/
+ etc/
+ examples/
+ httpd/
- keystone/
+ catalog/
+ common/
+ contrib/
+ identity/
+ locale/
- middleware/
__init__.py
auth_token.py
core.py
ec2_token.py
s3_token.py
swift_auth.py
+ openstack/
+ policy/
+ token/
__init__.py
clean.py
cli.py
config.py
exception.py
service.py
test.py
+ tests/
# WARRANTIES OR CONDITIONS OF ANY KIND, either expre
# License for the specific language governing permis
# under the License.
from keystone.common import serializer
from keystone.common import wsgi
from keystone import config
from keystone import exception
from keystone.openstack.common import jsonutils
CONF = config.CONF
# Header used to transmit the auth token
AUTH_TOKEN_HEADER = 'X-Auth-Token'
# Environment variable used to pass the request cont
CONTEXT_ENV = 'openstack.context'
# Environment variable used to pass the request para
PARAMS_ENV = 'openstack.params'
class TokenAuthMiddleware(wsgi.Middleware):
def process_request(self, request):
token = request.headers.get(AUTH_TOKEN_HEADE
context = request.environ.get(CONTEXT_ENV, {
context['token_id'] = token
request.environ[CONTEXT_ENV] = context
% 618 -: *NeoTree* Ne - 5.6k -: core.py Python FlyC 80+ yas-2- AC Git

Data Science Spaghetti **Code**

1.
Back to the basics

Code vs Software

Daniel Moisset: https://www.youtube.com/watch?v=4dlWg0B4ASw

"Code is something that runs on a Computer"

Code does not necessarily...

- Have tests
- Follow conventions
- Have documentation
- Follow processes

"Software is the programming text that is part of a deliverable"

You want to build Software...

- Maintainable
- Testable
- Deployable

Python is...

*Python is an interpreted, interactive, **object-oriented** programming language. It incorporates modules, exceptions, dynamic typing, very high level dynamic data types, and classes.*

Source: https://docs.python.org/3/faq/general.html#what-is-python

Object Oriented Programming (OOP)

*(OOP) is a programming paradigm based on the concept of "**objects**", which may **contain data**, in the form of fields, often known as **attributes**; and code, in the form of procedures, often known as **methods**.*

Source: https://en.wikipedia.org/wiki/Object-oriented_programming

How does an object look in Python?

Cookie Cutters & Cookies

Classes & Objects

```
class Cookie(object):
    def __init__(self, sugar=5):
        self.sugar = sugar
    def eat(self):
        pass
    def split(self):
        pass
```

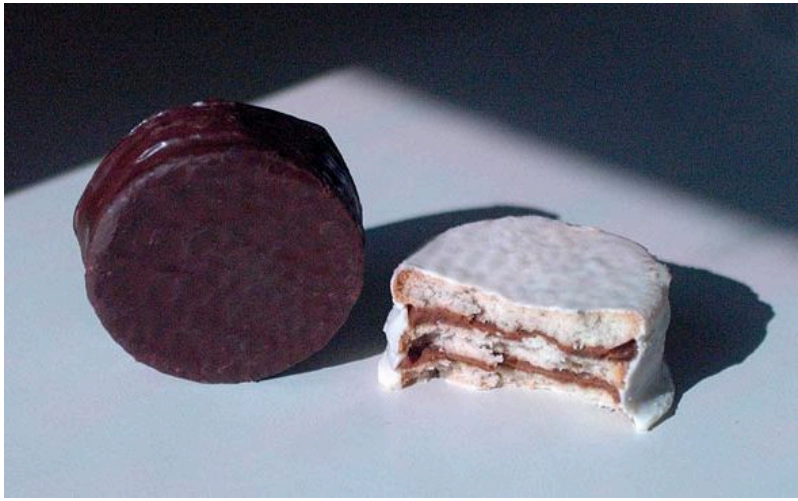
```
class Cookie(object):
    def __init__(self, sugar=5):
        self.sugar = sugar
    def eat(self):
        pass
    def split(self):
        pass
```

```
class Cookie(object):
    def __init__(self, sugar=5):
        self.sugar = sugar
    def eat(self):
        pass
    def split(self):
        pass
```

```
class Cookie(object):
    def __init__(self, sugar=5):
        self.sugar = sugar
    def eat(self):
        pass
    def split(self):
        pass
```

```
class Cookie(object):
    def __init__(self, sugar=5):
        self.sugar = sugar
    def eat(self):
        pass
    def split(self):
        pass

c = Cookie(3)
```

```
class Alfajor(Cookie):
  def __init__(self, chocolate=10, sugar=10):
      super(Alfajor, self).__init__(sugar=sugar)
      self.chocolate = chocolate

a = Alfajor(chocolate=20, sugar=30)
```



```
from sklearn import svm
data  = # multiple lines to load the data
X = # multiples lines extract the features
y = # ...
clf = svm.SVC()
clf.fit(X, y)
clf.predict(...)
# multiples lines store the results
```

How do I write good object oriented code?

How do I write good OO code?

- **DRY**
- **KISS**
- **SOLID**

DRY: Don't Repeat Yourself

Every piece of knowledge must have a single, unambiguous, authoritative representation within a system

Source: https://en.wikipedia.org/wiki/Don%27t_repeat_yourself

KISS: Keep it Simple Stupid

Simplicity is the ultimate sophistication

Leonardo Davicni

Be **SOLID**

- **S**ingle responsibility principle
- **O**pen/closed principle
- **L**iskov substitution principle
- **I**nterface segregation principle
- **D**ependency inversion principle

“Principles Of OOD”, Robert C. Martin Source: https://es.wikipedia.org/wiki/SOLID

Be **SOLID**

- **S**ingle responsibility principle
- **O**pen/closed principle
- **L**iskov substitution principle
- **I**nterface segregation principle
- **D**ependency inversion principle

“Principles Of OOD”, Robert C. Martin Source: https://es.wikipedia.org/wiki/SOLID

Learn OOP in Python

Coding Conventions

Coding Conventions

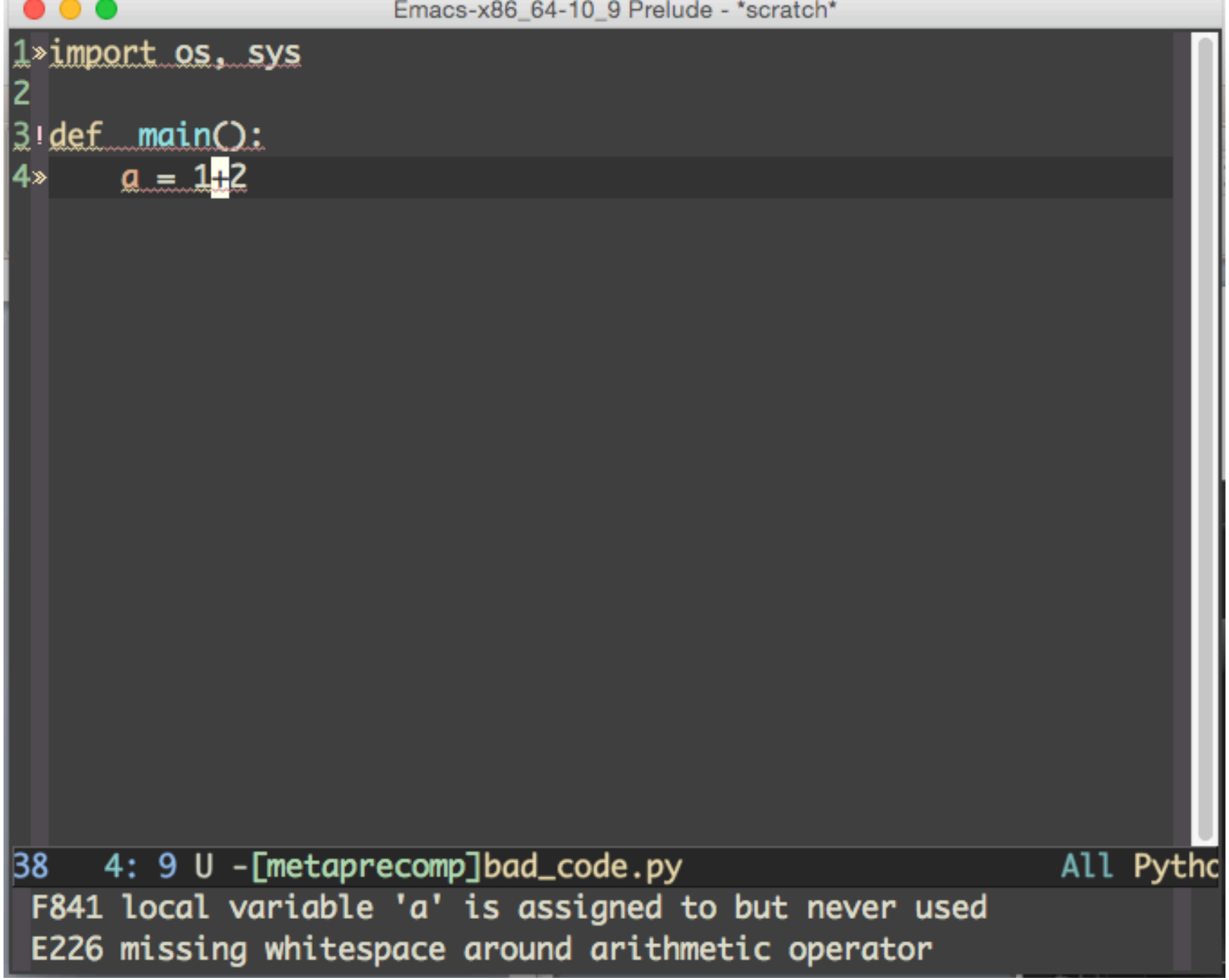
- “Readability counts” (PEP20)
- Spaces vs. Tabs
- Indentation rules
- Code organization
- PEP-8 is the de-facto style

PEP-8

```
# Aligned with opening delimiter.
foo = long_function_name(var_one, var_two,
                         var_three, var_four)
```

```
# Arguments on first line forbidden when not using vertical alignment.
foo = long_function_name(var_one, var_two,
    var_three, var_four)
```

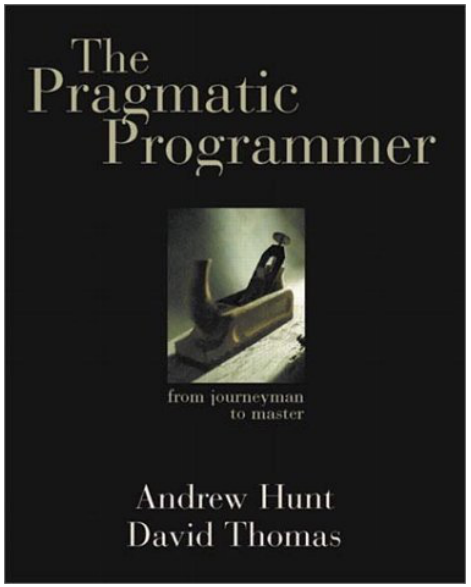
http://pep8.org

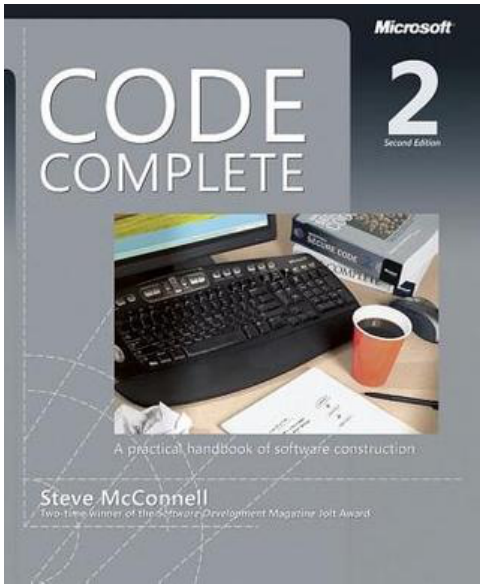

Emacs-x86_64-10_9 Prelude - *scratch*
1»import os, sys
2
3!def main():
4»   a = 1+2
38   4: 9 U -[metaprecomp]bad_code.py All Pythc
F841 local variable 'a' is assigned to but never used
E226 missing whitespace around arithmetic operator

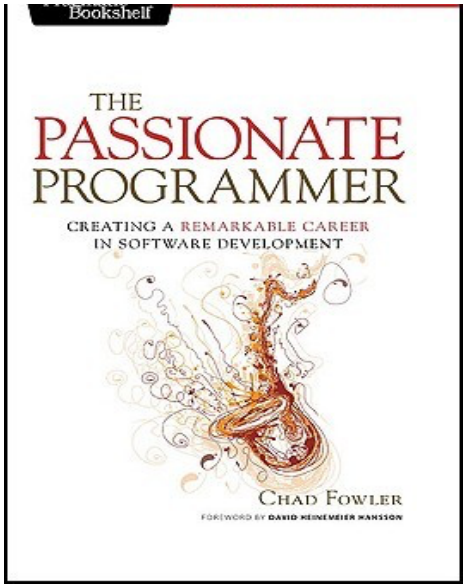
Other Topics

- Project structure
- Testing (Check out py.test!)
- Versioning and branching
- Code Reviews
- Software Development Life Cycle

Books!


The Pragmatic Programmer
from journeyman to master
Andrew Hunt
David Thomas


Microsoft
CODE COMPLETE
2
Second Edition
A practical handbook of software construction
Steve McConnell


Bookshelf
THE PASSIONATE PROGRAMMER
CREATING A REMARKABLE CAREER IN SOFTWARE DEVELOPMENT
CHAD FOWLER

Talks @ Europython

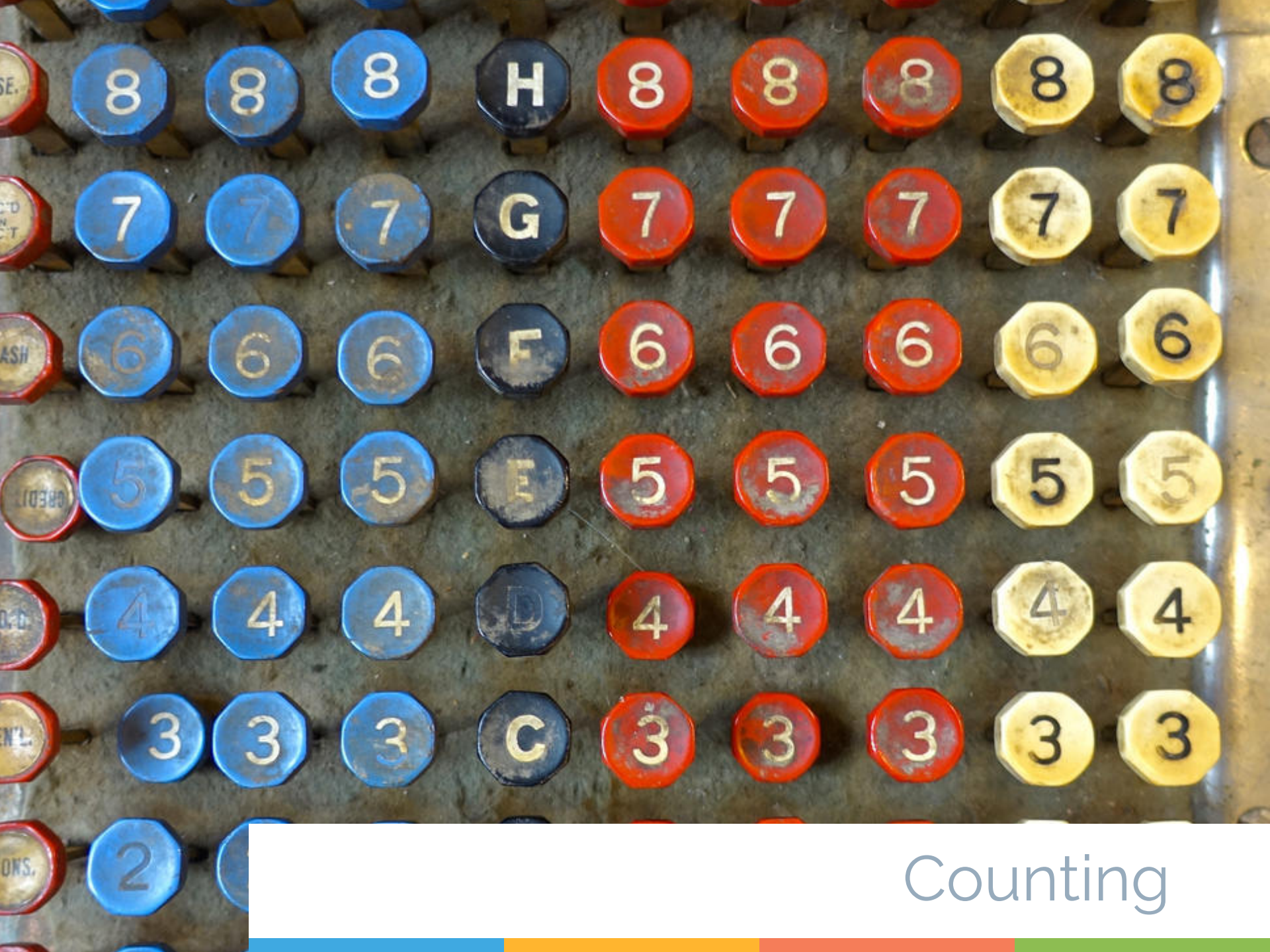
- Clean Code in Python by Mariano Anaya
  (Today at 15:45 Barria 2)
- What's the point of Object Orientation?
  by Iwan Vosloo (Thursday 11:15 A2)

2.
Tips & Tricks

2.

Tips & Tricks

The Collection Module


Counting

First Attempt

Using *dicts*!

```
items = ["a", "b", "a", "x", "x", "y", "c", "c",
"a"]

item_counts = {}

for i in items:
    if i in items:
        item_counts[i] = item_counts[i] + 1
    else:
        item_counts[i] = 1
```

Using *dicts* (*EAFP version*)

```
items = ["a", "b", "a", "x", "x", "y", "c", "c",
"a"]

item_counts = {}

for i in items:
    try:
        item_counts[i] = item_counts[i] + 1
    except KeyError:
        item_counts[i] = 1
```

We can do better

Let's use `defaultdict`

Using defaultdict

```
from collections import defaultdict

items = ["a", "b", "a", "x", "x", "y", "c", "c",
"a"]

item_counts = defaultdict(int)

for i in items:
    item_counts[i] = item_counts[i] + 1
```

```
defaultdict(int, {'a': 3, 'b': 1, 'c': 2, 'x': 2,
'y': 1})
```

Let's use `Counter`

Using Counter

```
from collections import Counter

items = ["a", "b", "a", "x", "x", "y", "c", "c",
"a"]

item_counts = Counter(items)
print(item_counts)
```

```
Counter({'a': 3, 'x': 2, 'c': 2, 'b': 1, 'y': 1})
```

`Counter`'s extra goodies

Extra goodies

```
>>> c = Counter(a=3, b=1)
>>> d = Counter(a=1, b=2)
>>> c.most_common()
>>> c.values()
>>> c + d
>>> c - d
>>> c & d          # intersection:  min(c[x],
d[x])
>>> c | d          # union:  max(c[x], d[x])
```

`Counter` is a class

Classes can be extended

Counter based PMF

```
from collections import Counter


class PMF(Counter):
    def normalize(self):
        total = float(sum(self.values()))
        for key in self:
            self[key] /= total
```

Counter based PMF

```
from collections import Counter


class PMF(Counter):
    def normalize(self):
        total = float(sum(self.values()))
        for key in self:
            self[key] /= total

    def __init__(self, *args, **kwargs):
        super(PMF, self).__init__(…)
        self.normalize()
```

On Counting and Python

- Use the and extend `Counter` class
- Awesome article from @TreyHunner on Counting [1].

[1] http://treyhunner.com/2015/11/counting-things-in-python/

Named Tuples

Named Tuples

- Code around the **`dict`**, `tuples` or **lists**
- Never know what to expect
- Code becomes hard to read

Example

```
from math import sqrt
pt1 = (1.0, 5.0)
pt2 = (2.5, 1.5)


line_length = sqrt((pt1[0] - pt2[0])**2 + (pt1[1]
- pt2[1])**2)
```

Source: http://stackoverflow.com/questions/2970608/what-are-named-tuples-in-python

Example

```
line_length = sqrt((pt1[0] - pt2[0])**2 + (pt1[1]
- pt2[1])**2)
```

Source: http://stackoverflow.com/questions/2970608/what-are-named-tuples-in-python

Example

```
from collections import namedtuple
from math import sqrt
Point = namedtuple('Point', 'x y')
pt1 = Point(1.0, 5.0)
pt2 = Point(2.5, 1.5)

line_length = sqrt((pt1.x - pt2.x)**2 + (pt1.y -
pt2.y)**2)
```

Source: http://stackoverflow.com/questions/2970608/what-are-named-tuples-in-python

It has cool methods

`_asdict`	Return a new OrderedDict which maps field names to their values
`_make(iterable)`	Class method that makes a new instance from an existing sequence or iterable.

Extending NamedTuple

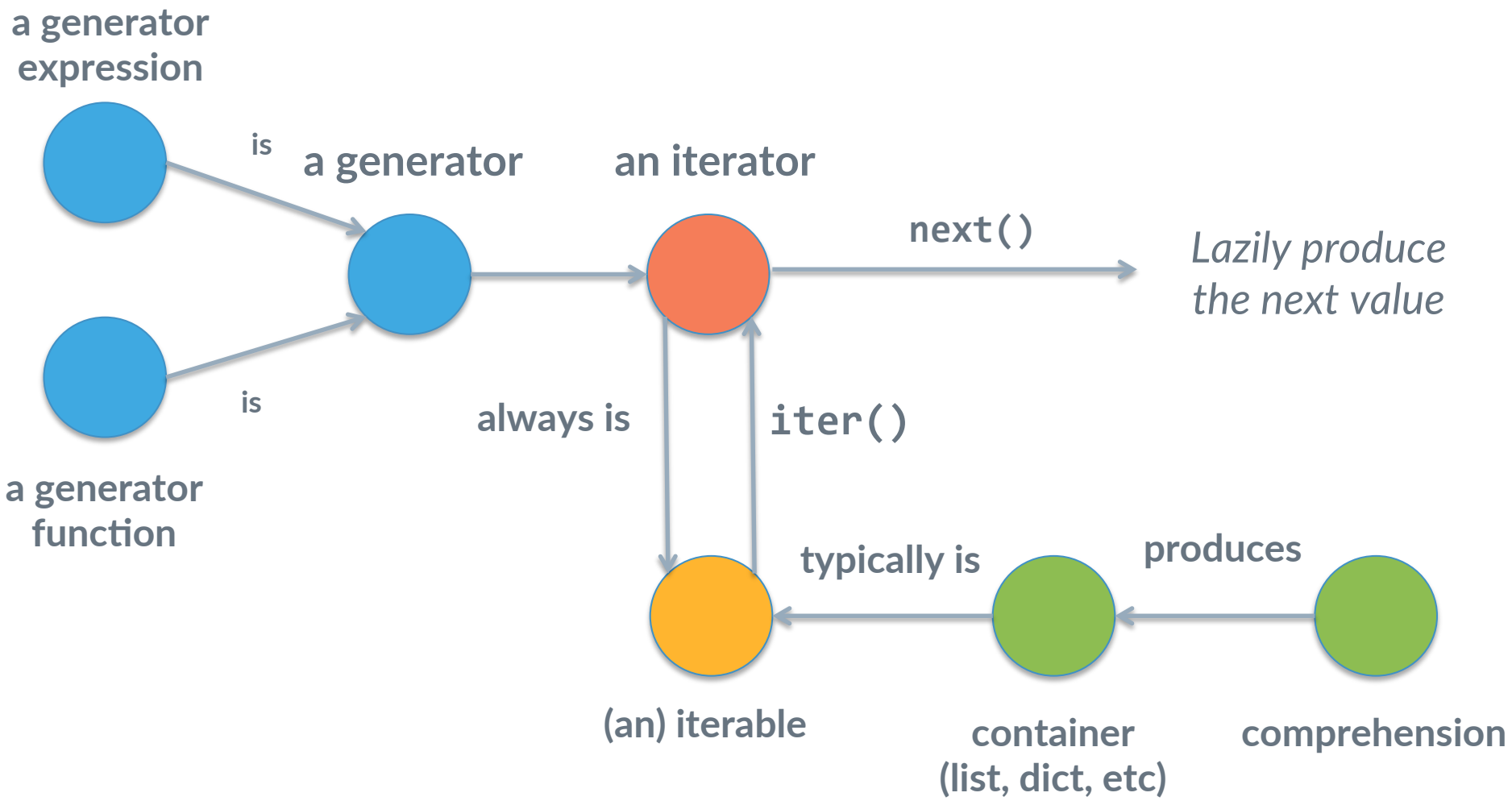
```
from collections import namedtuple

_HotelBase = namedtuple(
      'HotelDescriptor',
      ['cluster_id', 'trust_score', 'reviews_count',
       'category_scores', 'intensity_factors'],
)

class HotelDescriptor(_HotelBase):
    def compute_prior(self):
        if not self.trust_score or not self.reviews_count:
            raise NotEnoughDataForRanking("…")
        return _compute_prior(self.trust_score,…)
```

2.

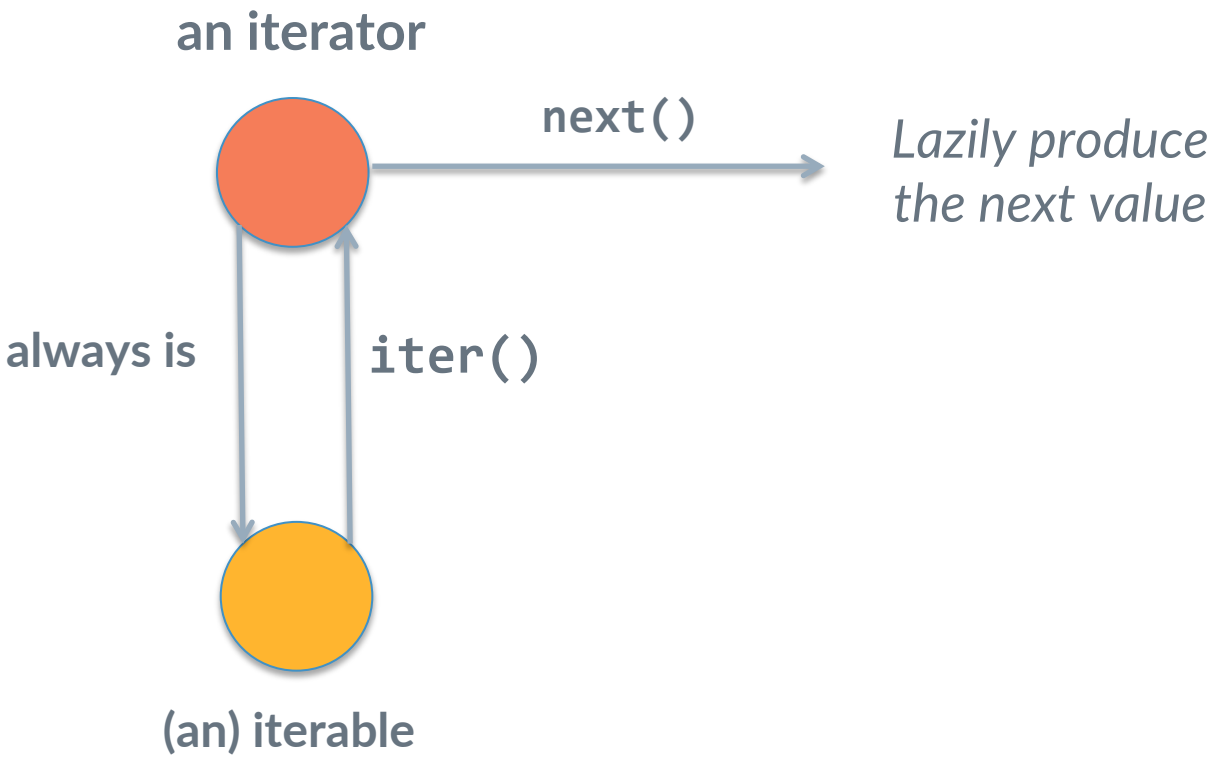
Tips & Tricks

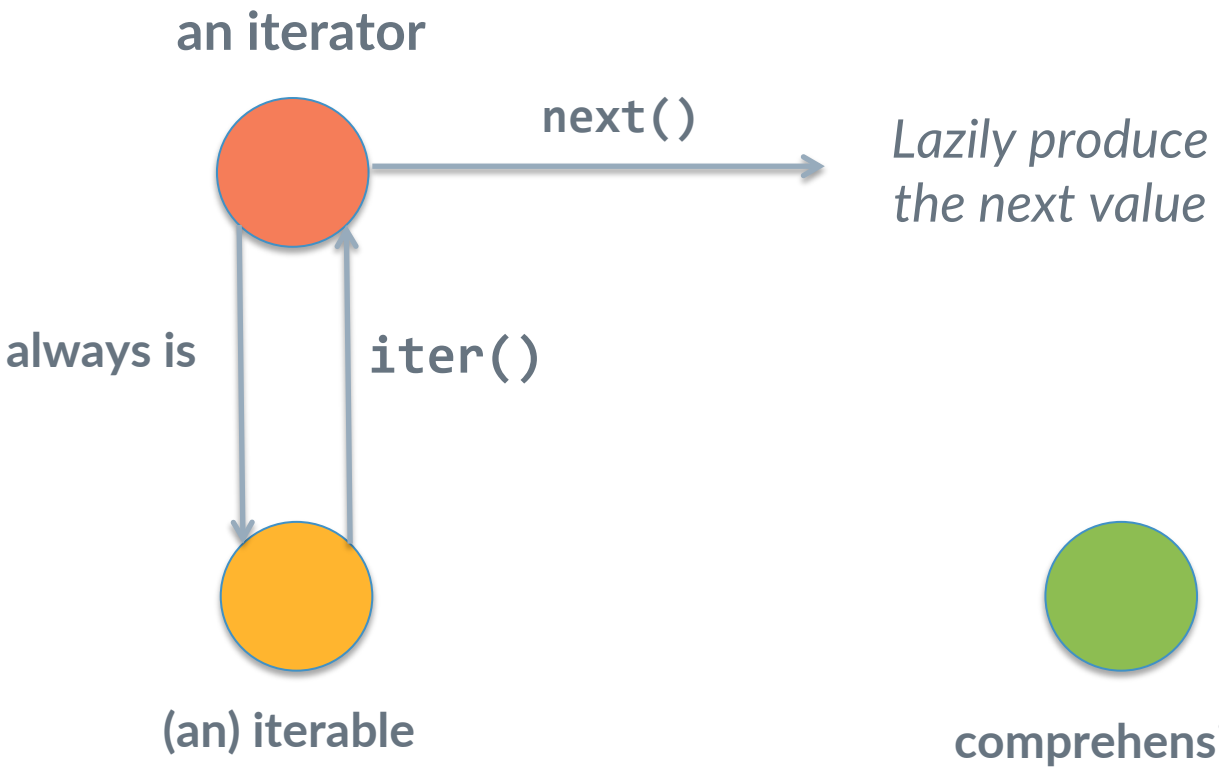
2.1 Iterators & Iterables

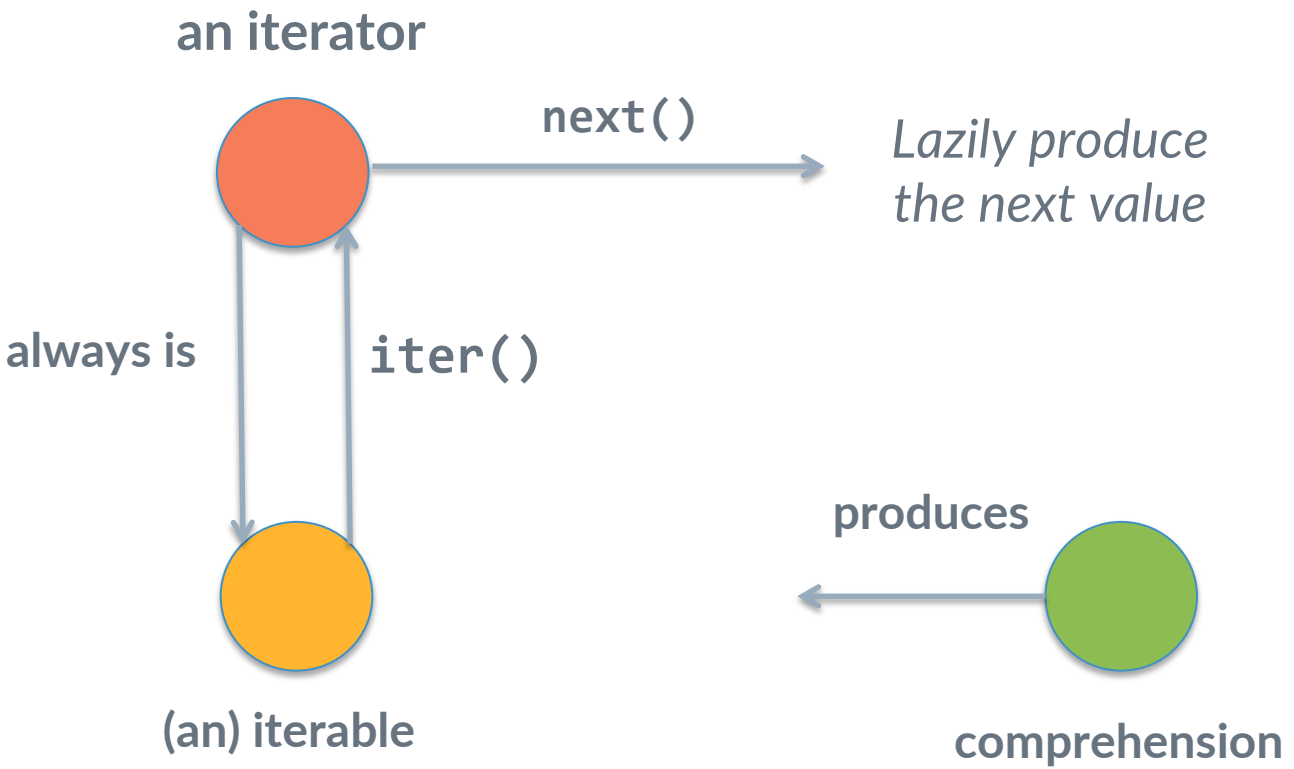
```
l = [1, 2, 3, 4]
for i in x:
    print(x)
```

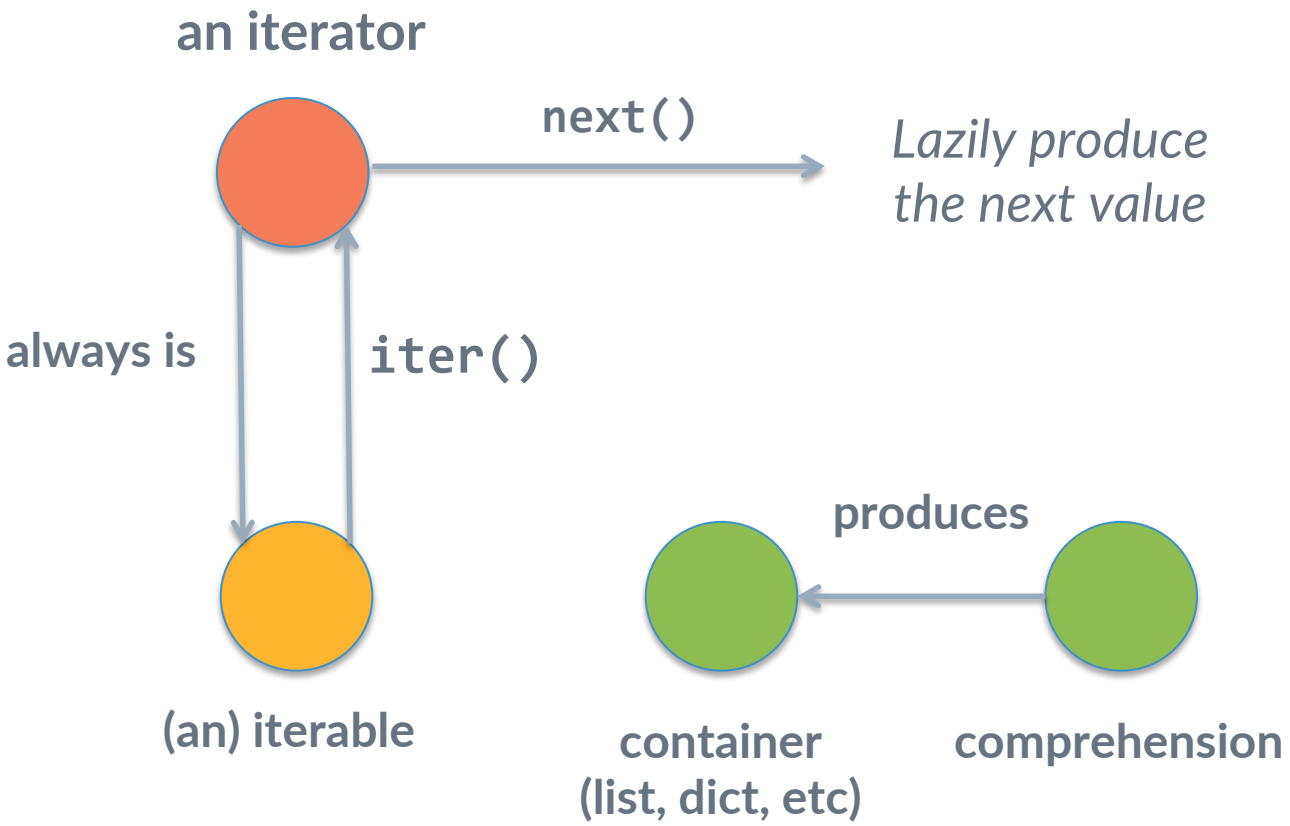


By Vincent Driessen - Source: http://nvie.com/posts/iterators-vs-generators/

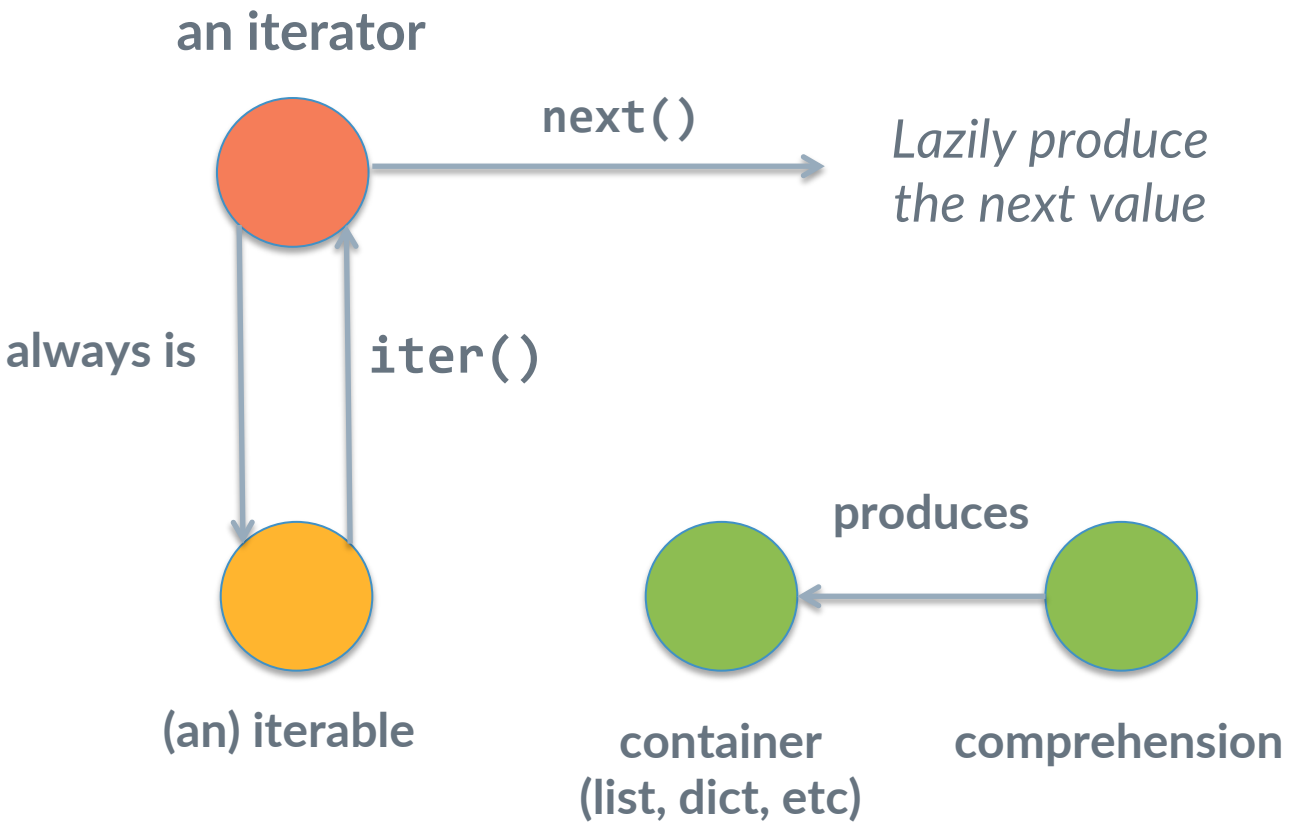

an iterator
next()
Lazily produce
the next value
always is
iter()
(an) iterable


an iterator
next()
Lazily produce
the next value
always is
iter()
(an) iterable
comprehension

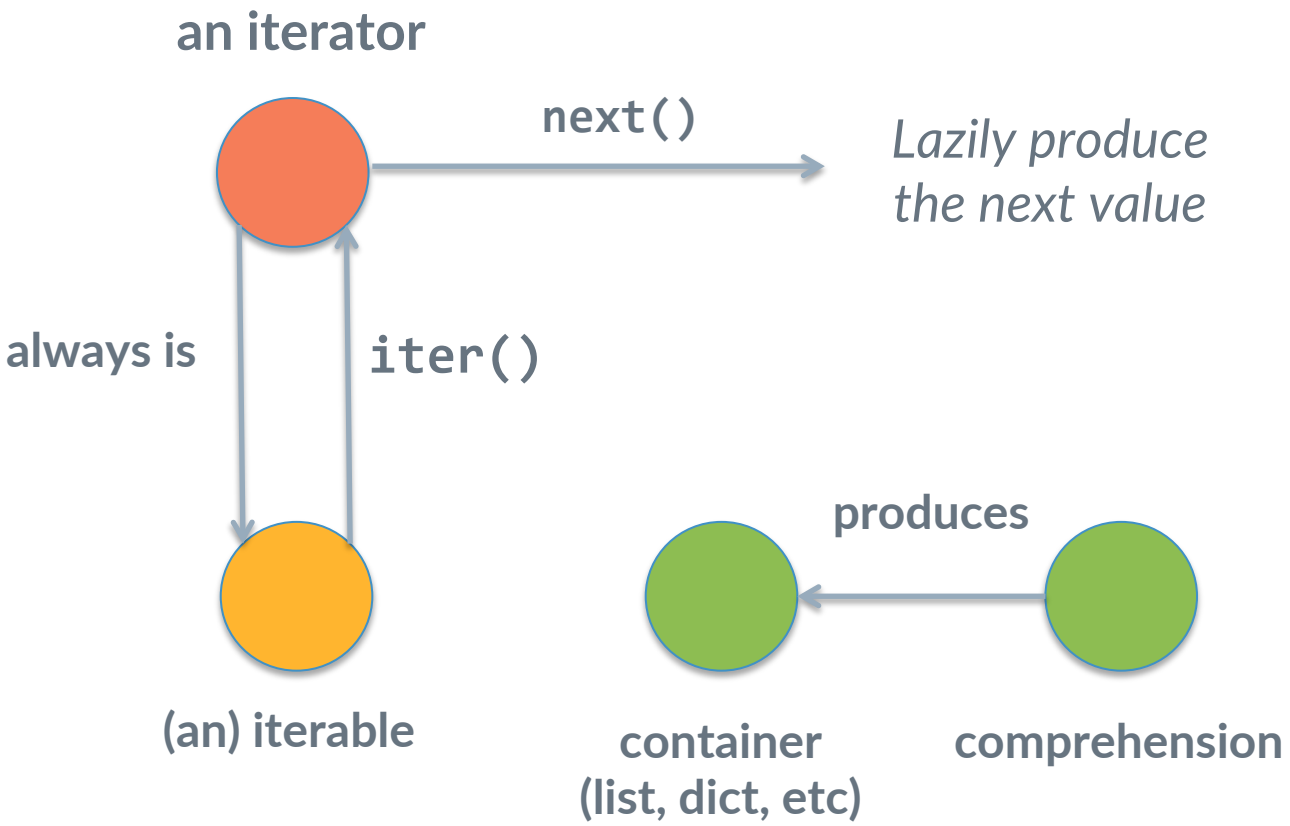

an iterator
next()
Lazily produce
the next value
always is
iter()
produces
(an) iterable
comprehension

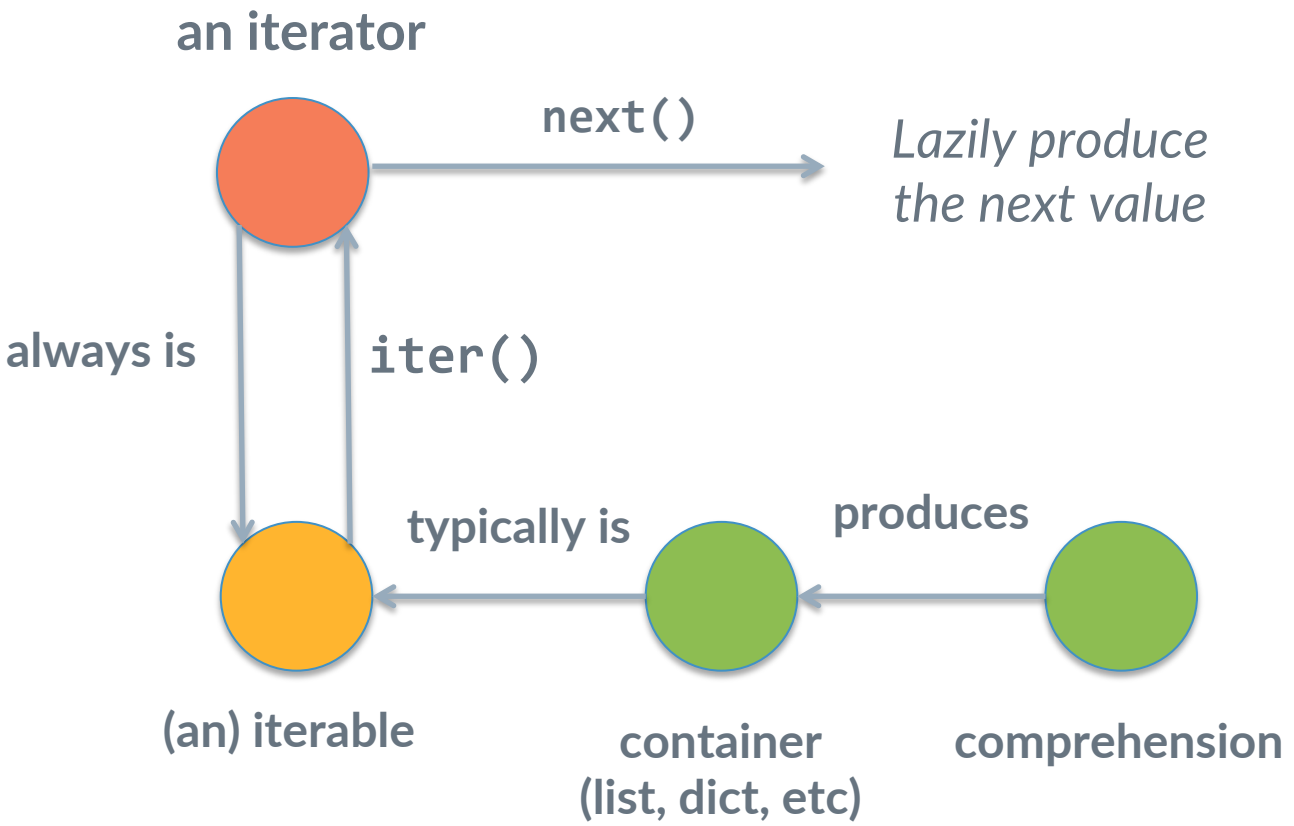

an iterator
next()
Lazily produce
the next value
always is
iter()
produces
(an) iterable
container
(list, dict, etc)
comprehension

```
assert 1 in [1, 2, 3]
```

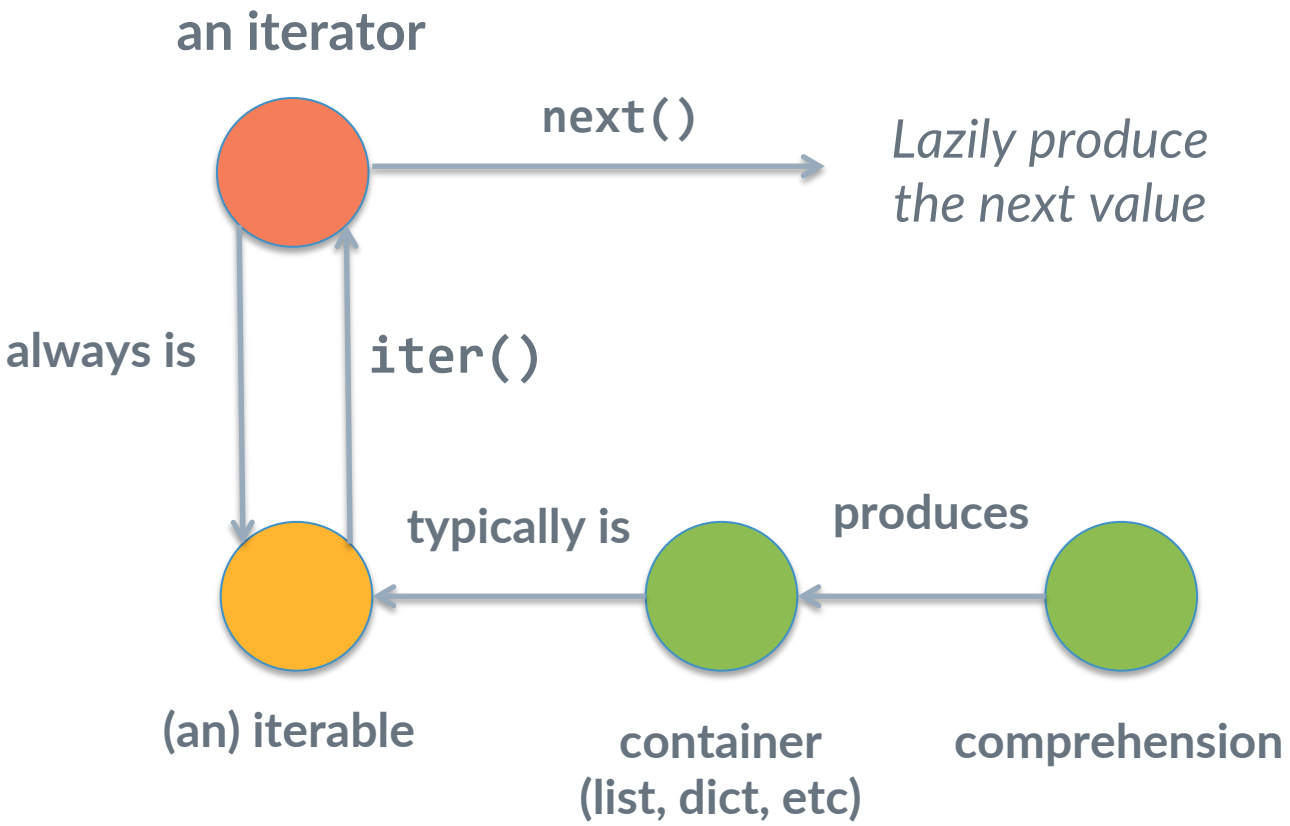


```
assert 1 in {1, 2, 3}
```

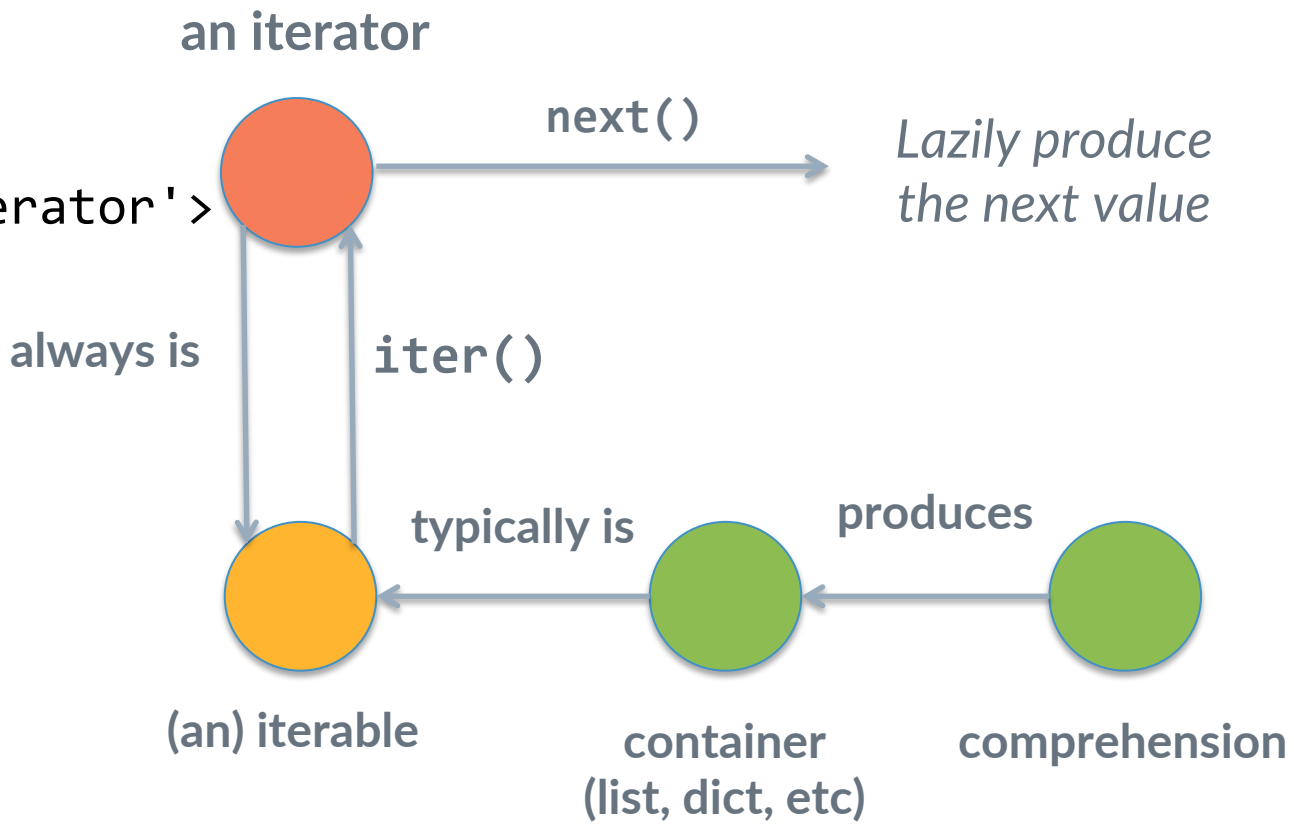



an iterator
next()
Lazily produce
the next value
always is
iter()
typically is
produces
(an) iterable
container
(list, dict, etc)
comprehension

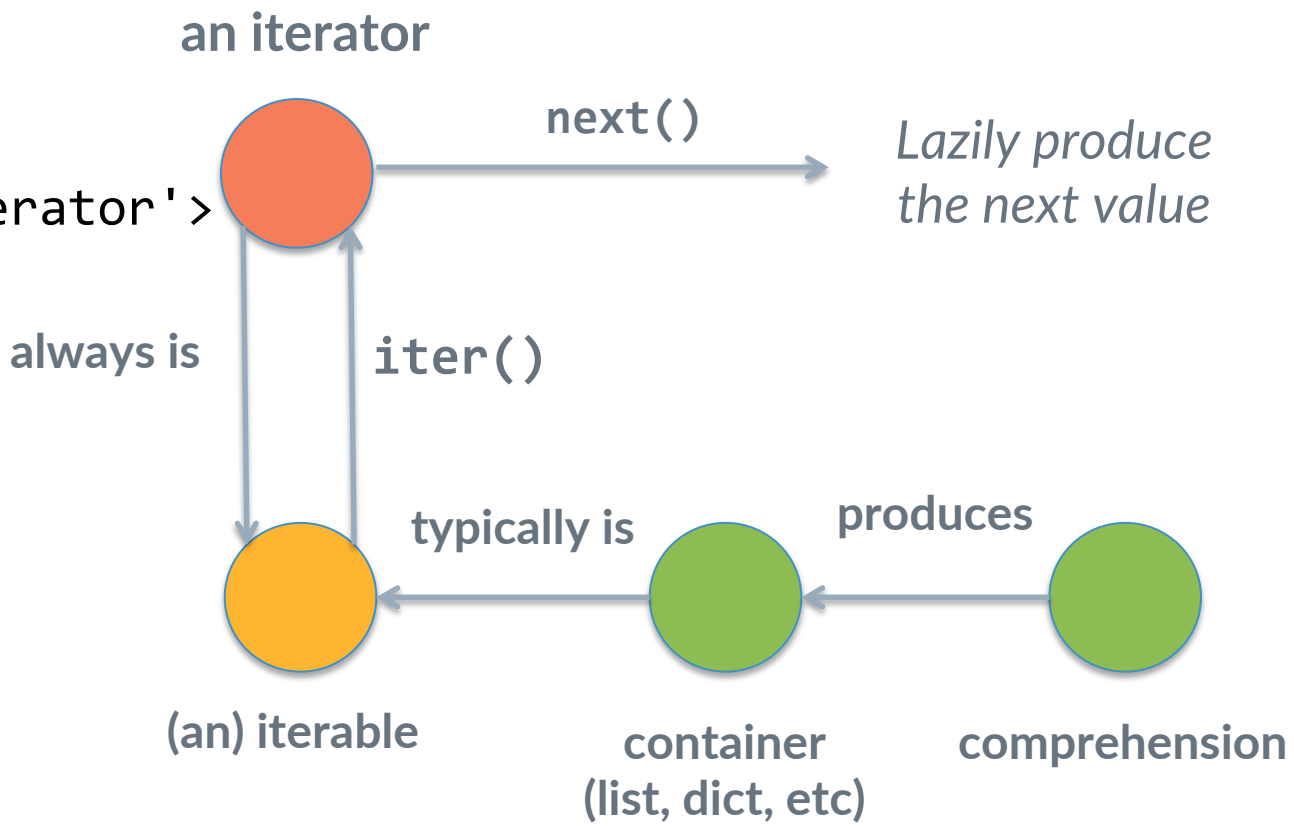
```
l = [1, 2, 3, 4]
x = iter(l)
y = iter(l)
```



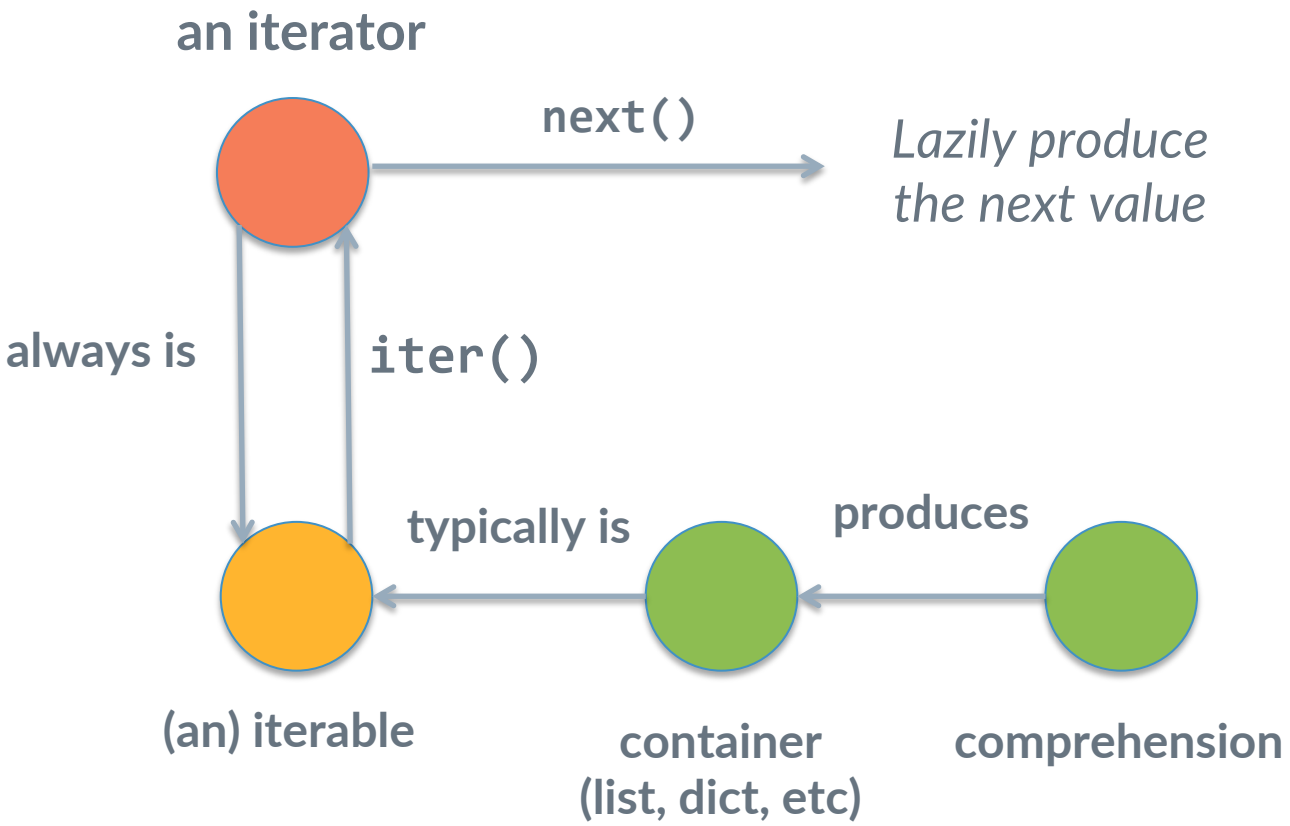
```
l = [1, 2, 3, 4]
x = iter(l)
y = iter(l)

type(l)
>> <class 'list'>
type(x)
>> <class 'list_iterator'>
```



```
l = [1, 2, 3, 4]
x = iter(l)
y = iter(l)

type(l)
>> <class 'list'>
type(x)
>> <class 'list_iterator'>

next(x)
>> 1
next(y)
>> 1
next(y)
>> 2
```



```
l = [1, 2, 3, 4]
for e in l:
    print(e)
```

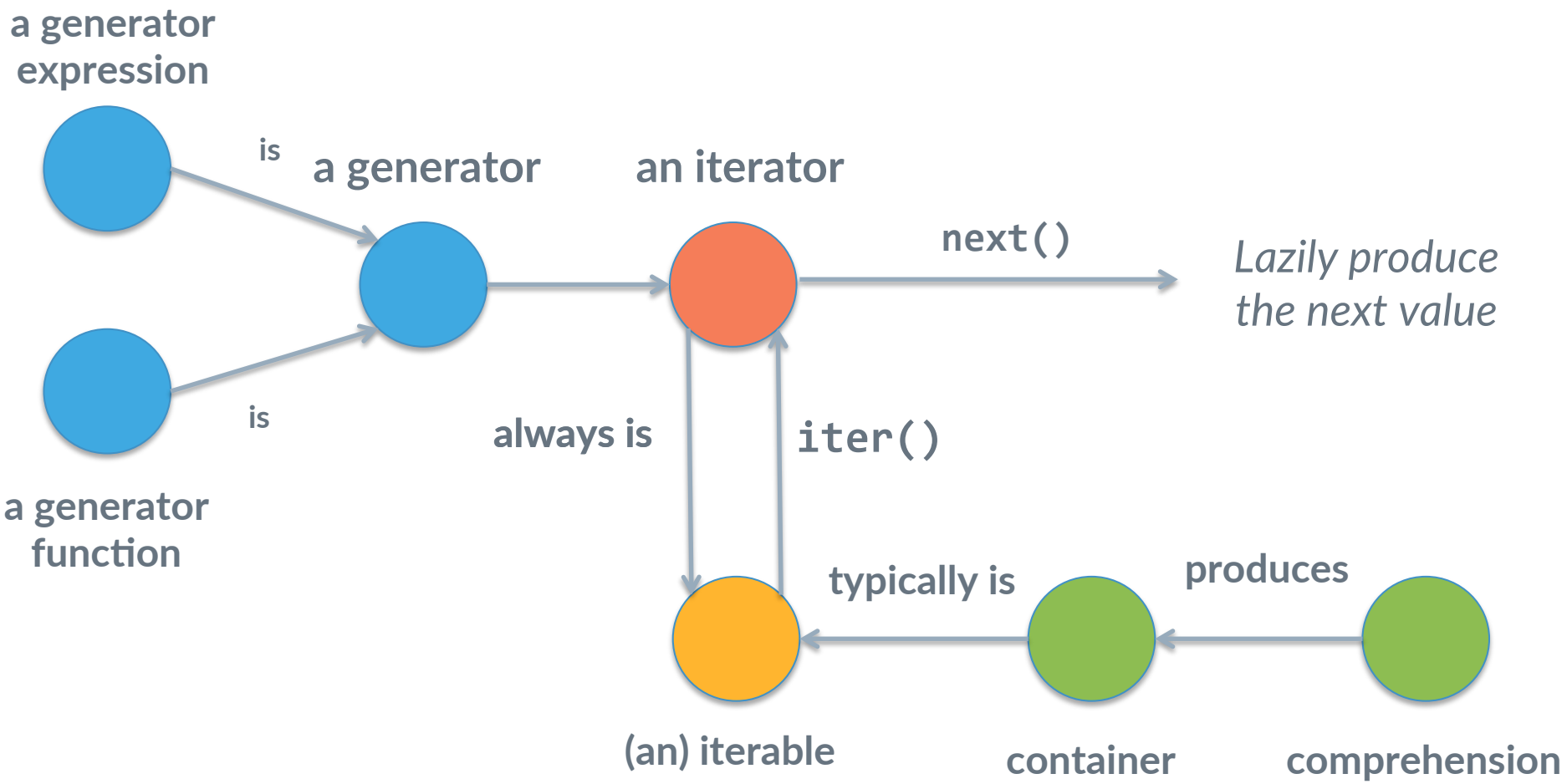


Iterables

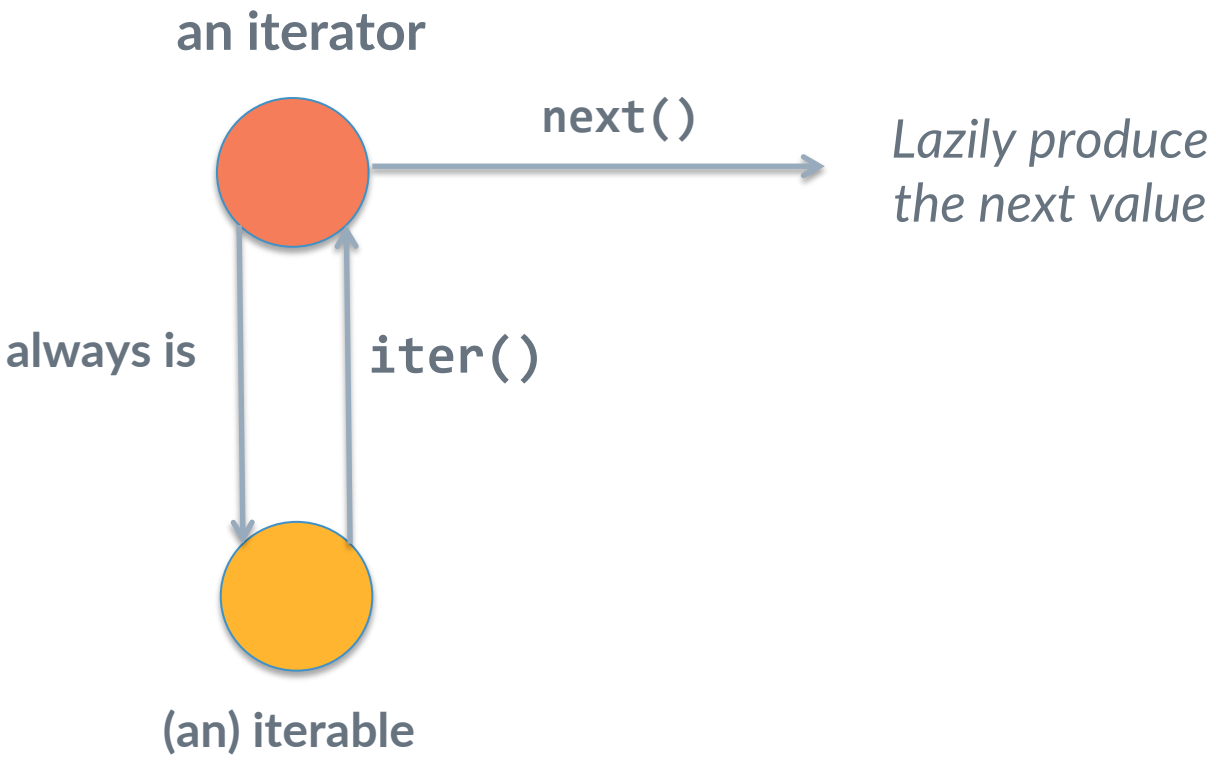
- An ***iterable*** is any object that can return an ***iterator***
- Containers, files, sockets, etc.
- Implement `__iter__()`.
- Some of them may be infinite
- The `itertools` contain many helper functions

```
class InverseReader(object):
    def __init__(self):
        with open('file.txt') as f:
            self.lines = f.readlines()
        self.index = len(self.lines) - 1

    def __iter__(self):
        return self

    def next(self):  # Python 3 __next__
        self.index -= 1
        if self.index < 0:
            raise StopIteration
        return self.lines[self.index]
```

```
ir = InverseReader()

for line in ir:
    print line
```

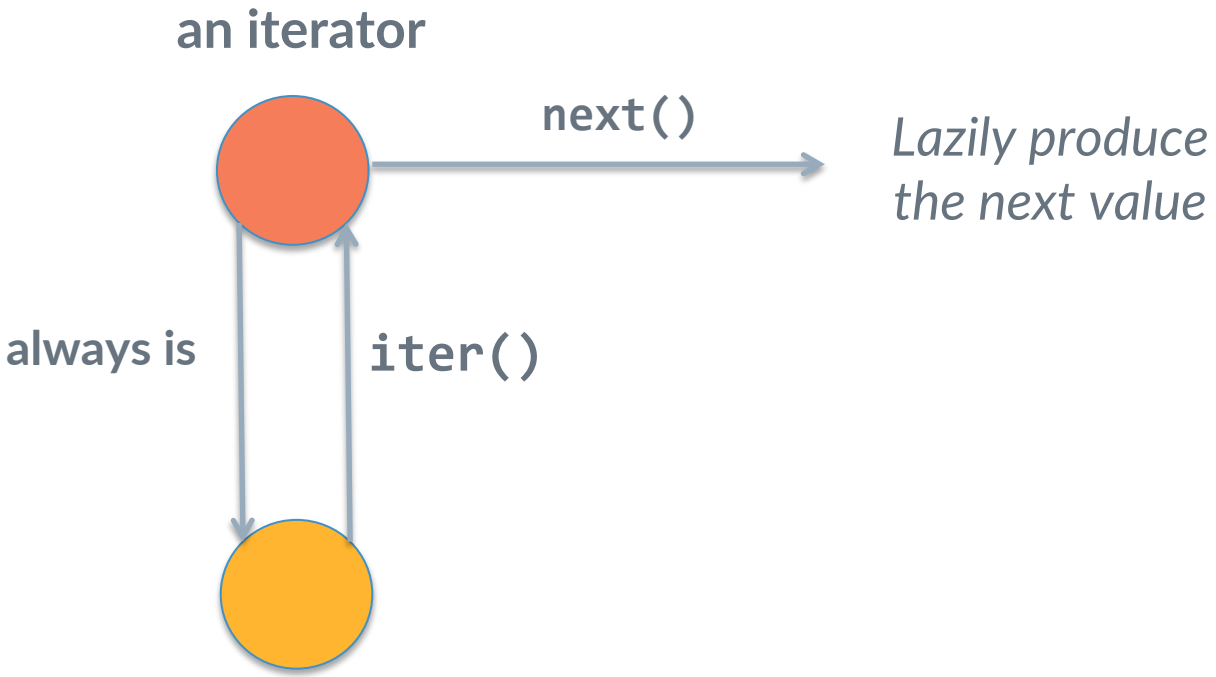

a generator expression
is
a generator
an iterator
next()
Lazily produce the next value
is
a generator function
always is
iter()
typically is
produces
(an) iterable
container
comprehension

a generator expression



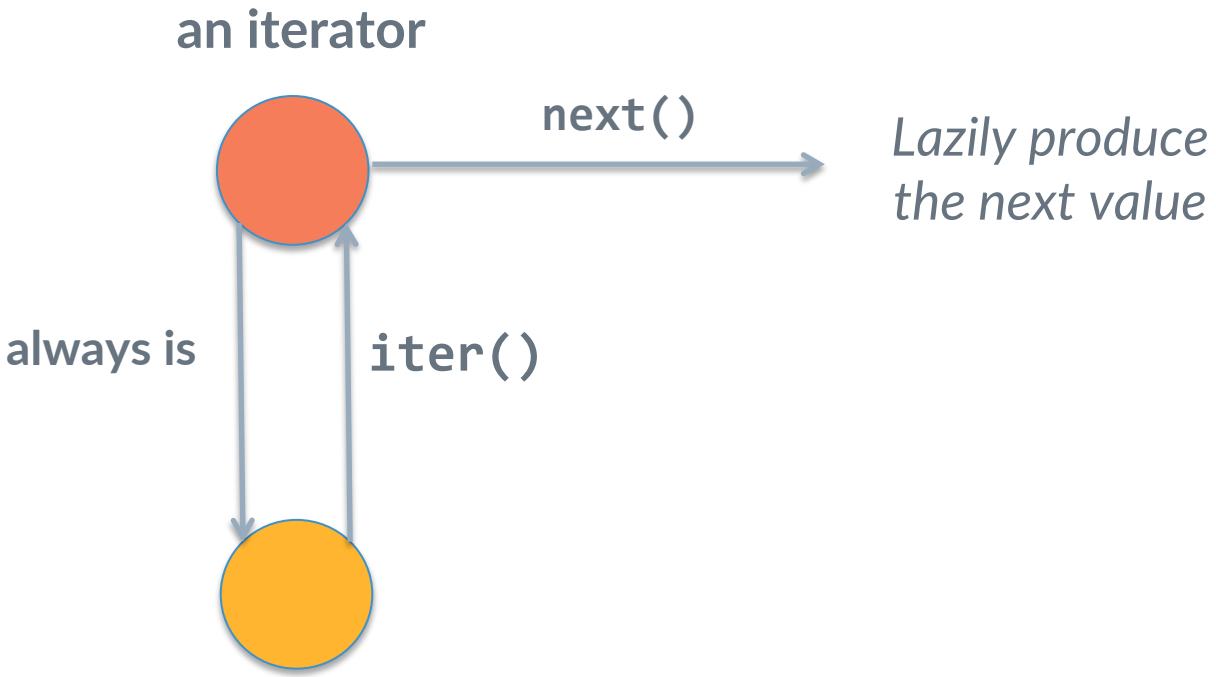
a generator expression

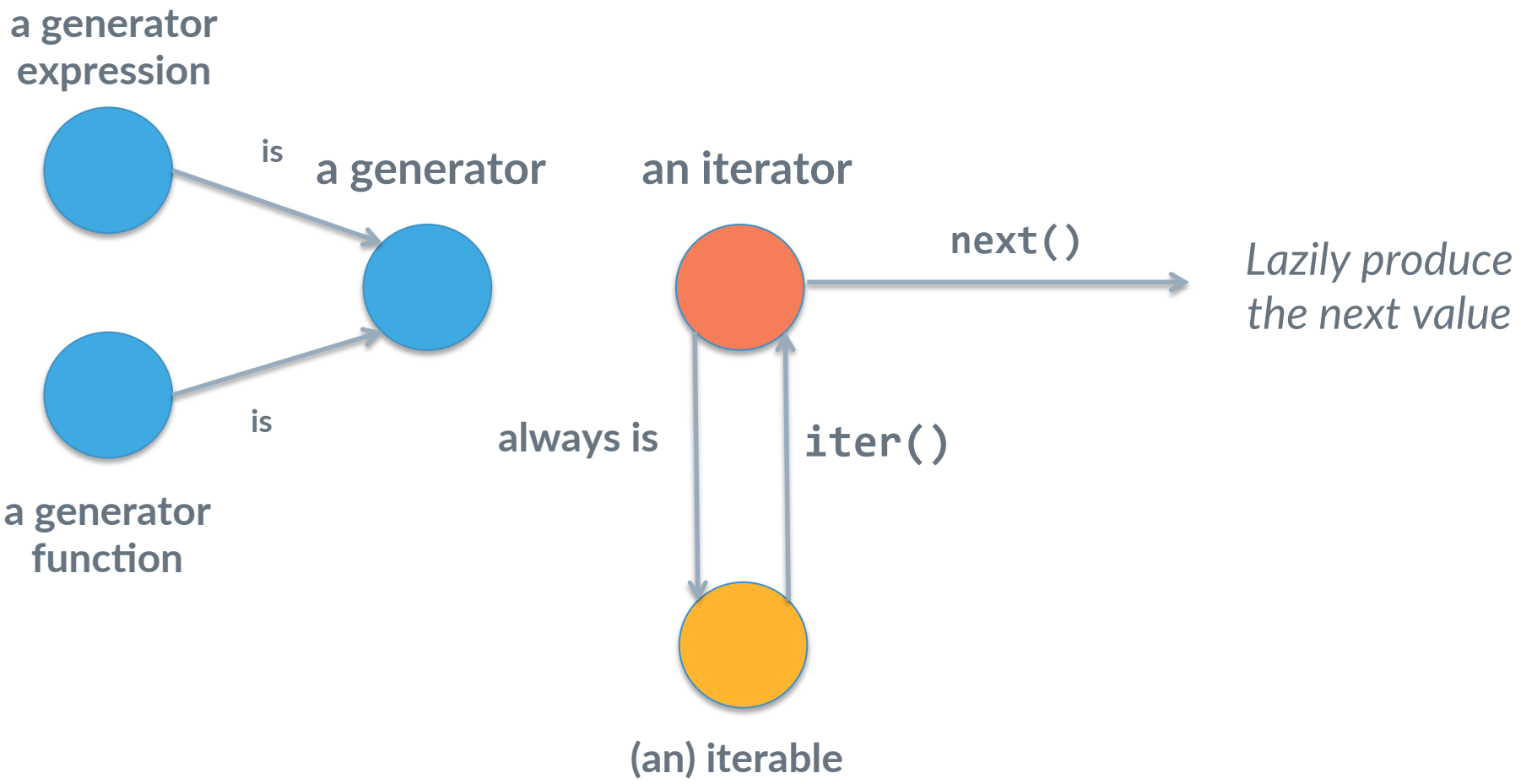
a generator function

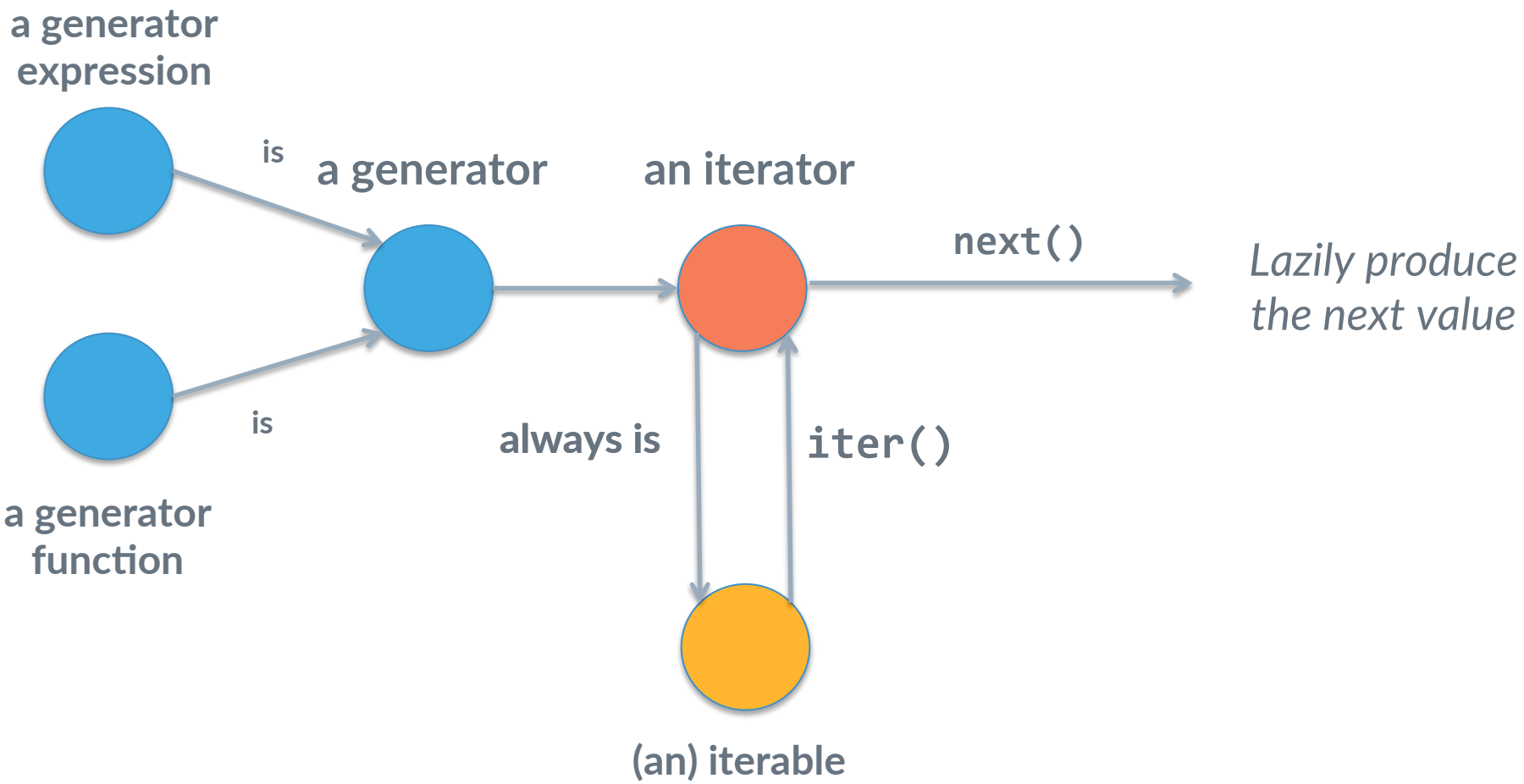


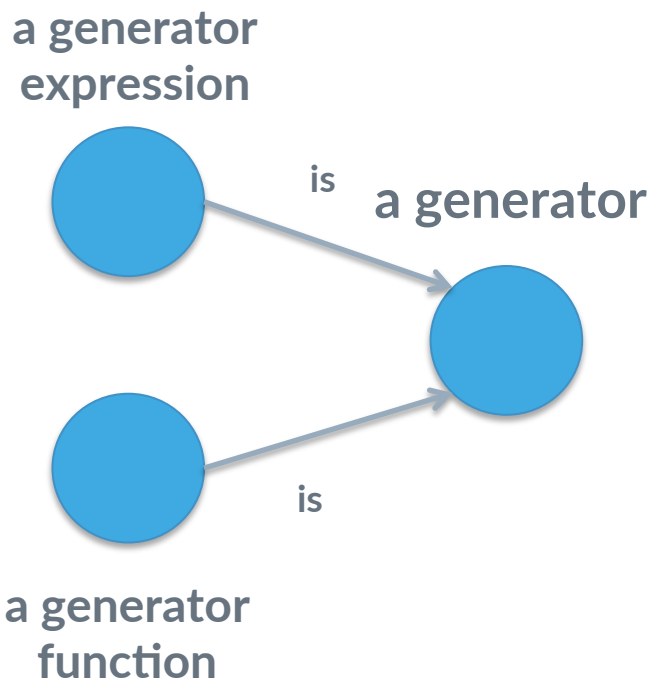
a generator expression

a generator function

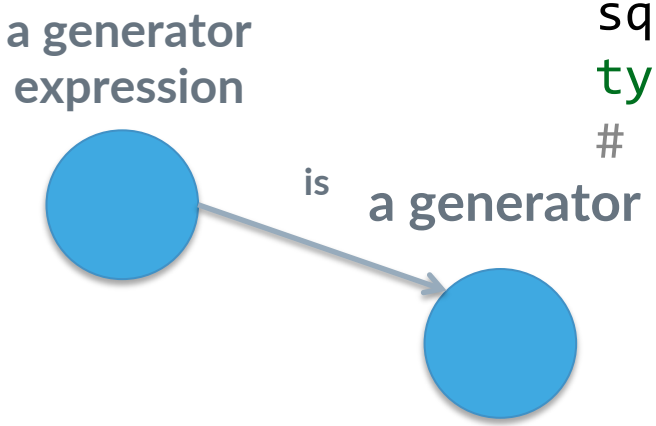


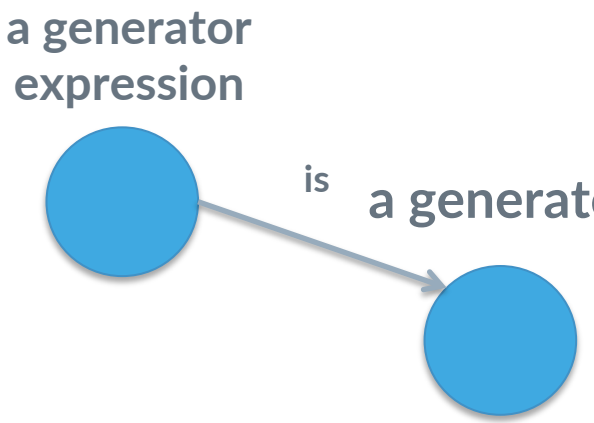

a generator expression
is
a generator
an iterator
next()
Lazily produce the next value
is
always is
iter()
a generator function
(an) iterable


a generator expression
is
a generator
an iterator
next()
Lazily produce the next value
is
a generator function
always is
iter()
(an) iterable

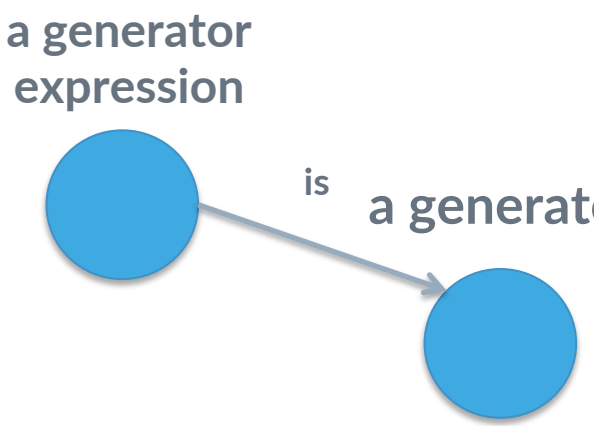

a generator expression
is
a generator
is
a generator function

```
numbers = [x for x in range(1, 10)]
squares = [x * x for x in numbers]
type(squares)
# list
```

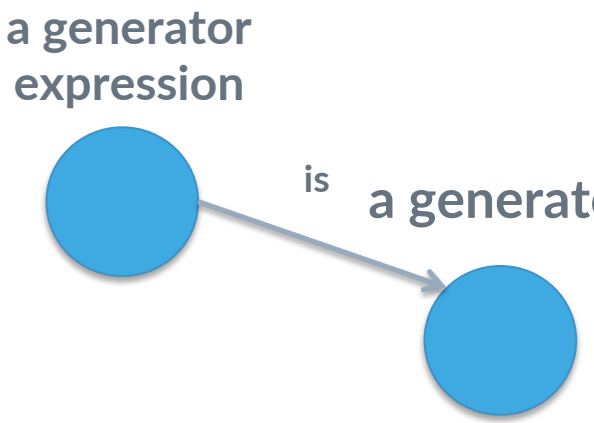





```
numbers = [x for x in range(1, 10)]
squares = [x * x for x in numbers]
type(squares)
# list

lazy_squares = (x * x for x in numbers)
lazy_squares
# <generator object <genexpr> at
0x104c6da00>
```



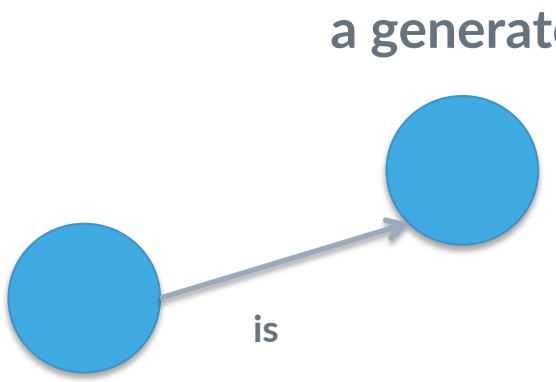

```
numbers = [x for x in range(1, 10)]
squares = [x * x for x in numbers]
type(squares)
# list

lazy_squares = (x * x for x in numbers)
lazy_squares
# <generator object <genexpr> at
0x104c6da00>

next(lazy_squares)
# 1
next(lazy_squares)
# 4
x
```



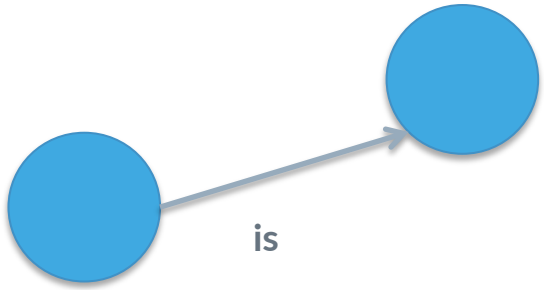

```
numbers = [x for x in range(1, 10)]
squares = [x * x for x in numbers]
type(squares)
# list
```

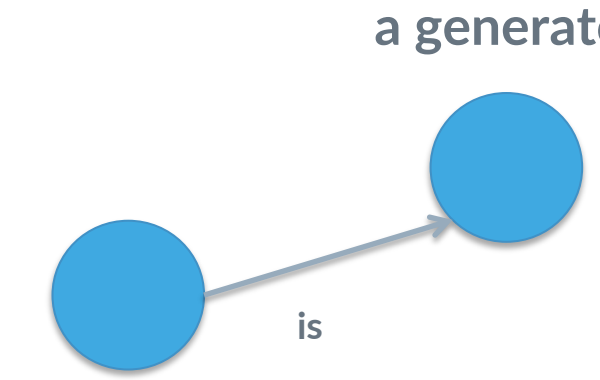
```
lazy_squares = (x * x for x in numbers)
lazy_squares
# <generator object <genexpr> at
0x104c6da00>

next(lazy_squares)
# 1
next(lazy_squares)
# 4
```

```
lazy_squares = (x * x for x in numbers)
for x in lazy_squares:
        print x
```

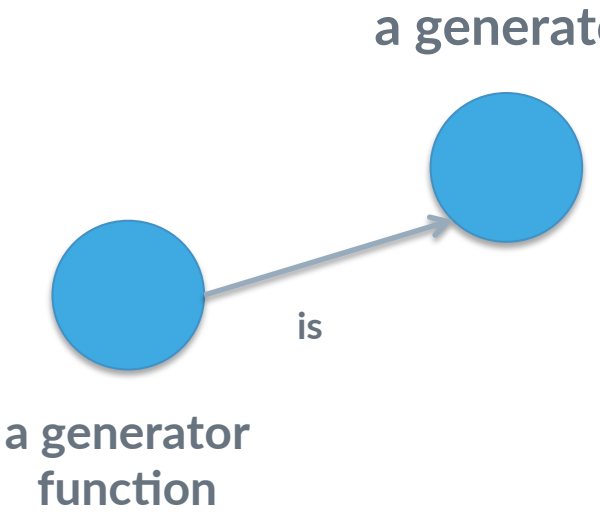

a generator
is
a generator
function

```
def fib():
    prev, curr = 0,1
    while True:
        yield curr
        prev,curr = curr, prev+curr
```





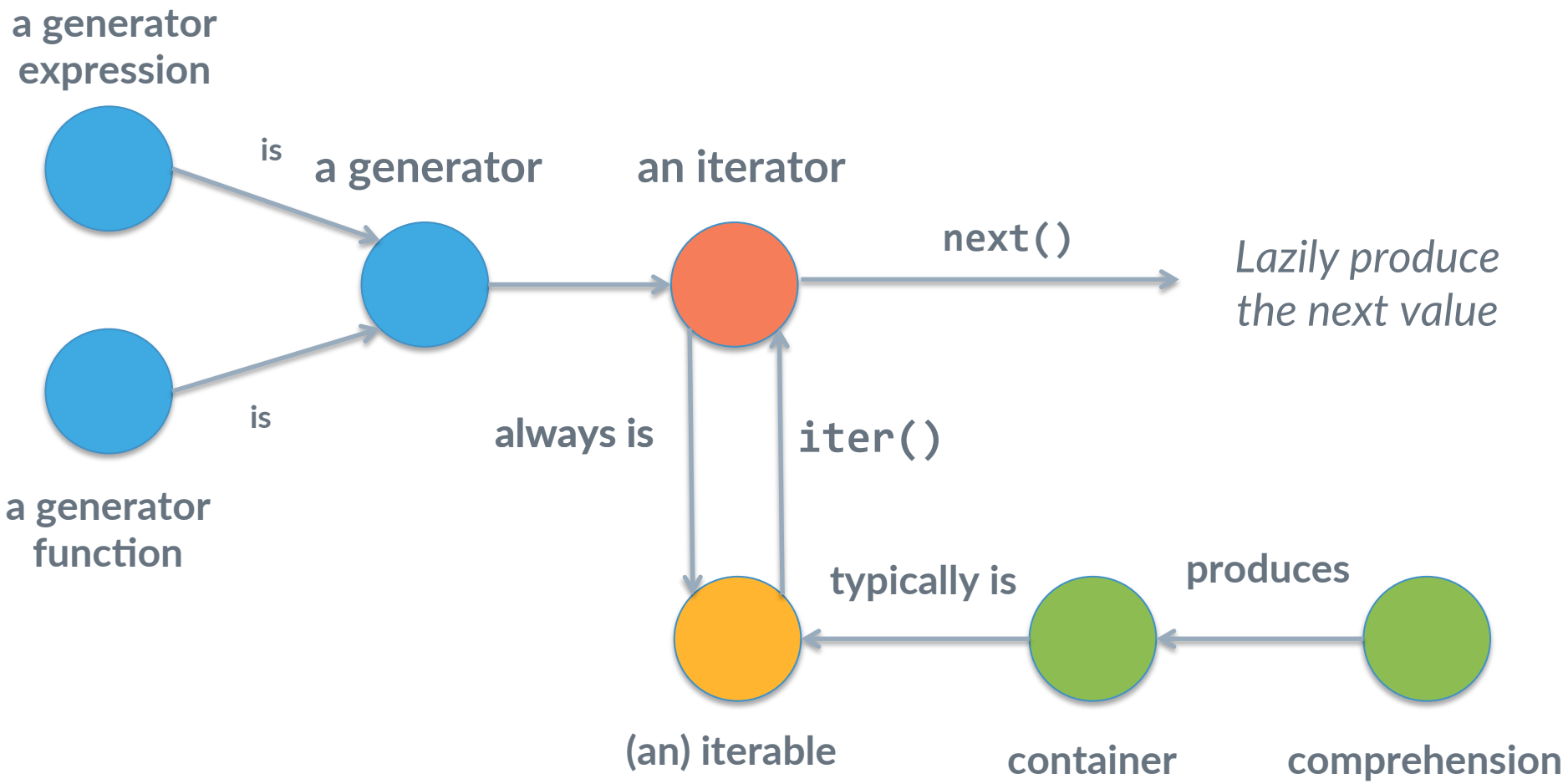

```
def fib():
    prev, curr = 0,1
    while True:
        yield curr
        prev,curr = curr, prev+curr

f = fib()

next(f)
# 1
next(f)
# 1
next(f)
# 2
```




```
def fib():
    prev, curr = 0,1
    while True:
        yield curr
        prev,curr = curr, prev+curr
```

```
for x in islice(fib(), 0,3):
    print x
# 1
# 1
# 2
```

```
class HdfsLineSentence(object):
    def __init__(self, source):
        self.source = source

    def __iter__(self):
        stream = self.source.open('r')
        for line in stream:
            cid, s = line.split('\t')
            # decode and do some work with s
            yield s

sentences = HdfsLineSentence(...)
for s in setences:
    print(s)
```


a generator expression
is
a generator
an iterator
next()
Lazily produce the next value
is
a generator function
always is
iter()
typically is
produces
(an) iterable
container
comprehension

What does it have to do with Data Processing?

Unknown amout of data

Not enough memory

Data streaming via lazy evaluation

Data processing pipelines through iterables

Chaining iterables

```
class HdfsLineSentence(object):
    def __init__(self, source):
        self.source = source

    def __iter__(self):
        stream = self.source.open('r')
        for line in stream:
            cid, s = line.split('\t')
            # decode and do some work with s
            yield s

sentences = HdfsLineSentence(...)
for s in sentences:
    print s
```

```
class FilterComment(object):
    def __init__(self, source):
        self.source = source

    def __iter__(self):
        for s in self.source:
            if s[0] != "#":
                yield s
```

```
class FilterComment(object):
    def __init__(self, source):
        self.source = source

    def __iter__(self):
        for s in self.source:
            if s[0] != "#":
                yield s


sents = FilterComment(HdfsLineSentence(source))
for s in sents:
      print s
```

```
class FilterComment(object):
    def __init__(self, source):
        self.source = source

    def __iter__(self):
        for s in self.source:
            if s[0] != "#":
                yield s


sents = FilterComment(HdfsLineSentence(source))
for s in sents:
      print s
```

```
class FilterComment(object):
    def __init__(self, source):
        self.source = source

    def __iter__(self):
        for s in self.source:
            if s[0] != "#":
                yield s


sents = FilterComment(HdfsLineSentence(source))
for s in sents:
      print s
```

```
def filter_comment(source):
    for s in source:
            if s[0] != "#":
                yield s

sents = filter_comment(HdfsLineSentence(source))

for s in sents:
     print s
```

Talks @ Europython

- Iteration, iteration, iteration by John Sutherland (Friday 15:45 Barria 1)

3.
Conclusions

Data Scientists / Engineers / ML Developers should learn...

`collections` and `itertools` modules

Iterables and iterators for data processing pipelines

Object oriented programming

Good software engineering practices

Credits

Counting things in Python: http://treyhunner.com/2015/11/counting-things-in-python/

PMF Class based on Vik Paruchuri's https://www.dataquest.io/blog/python-counter-class/

Ideas from Iterables taken from RaRe Technologies blog; http://rare-technologies.com/data-streaming-in-python-generators-iterators-iterables/

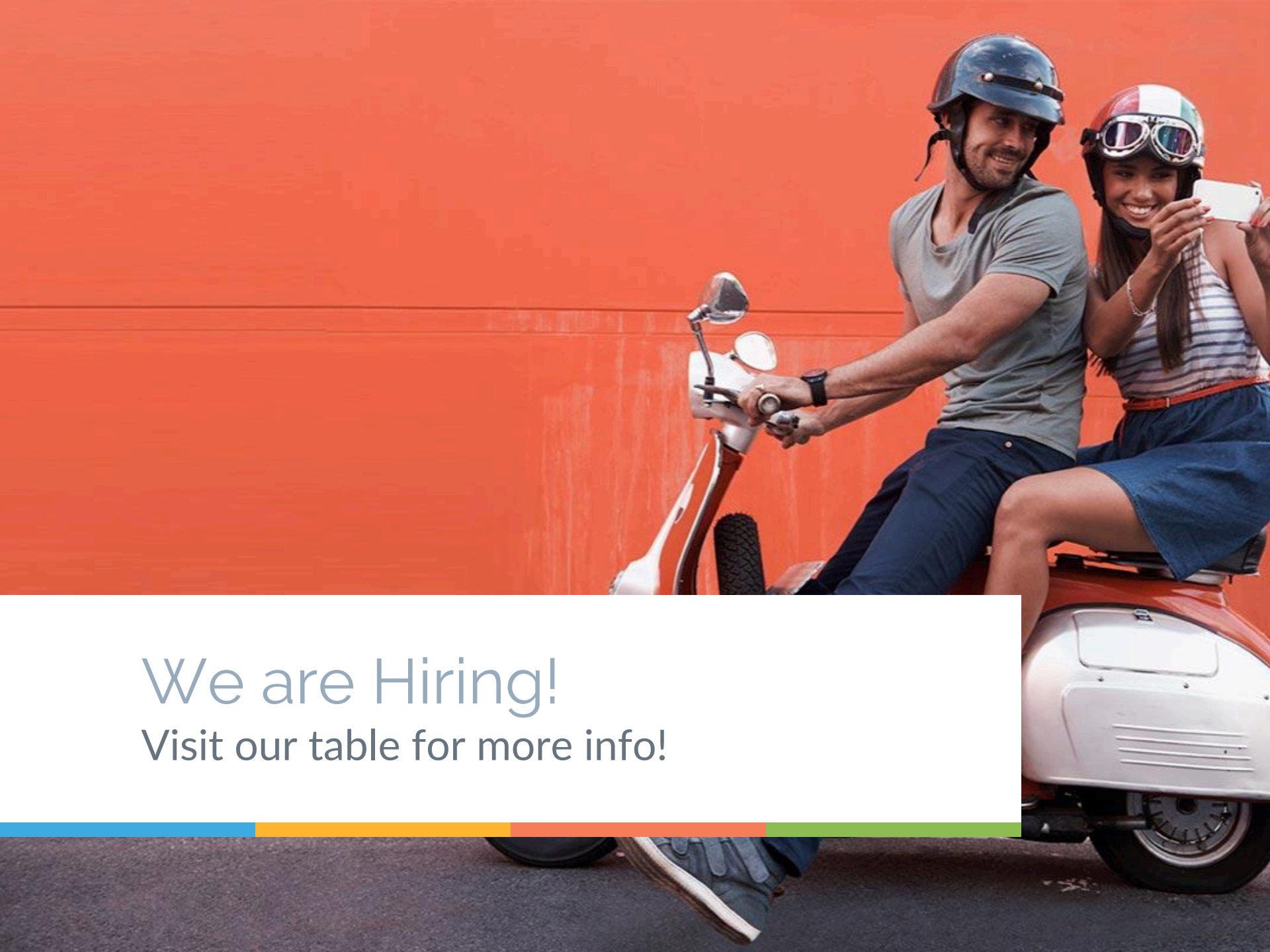
Iterators and Iterables based on work of Vincent Driessen : http://nvie.com/posts/iterators-vs-generators/

Spaguetti: https://www.flickr.com/photos/129610671@N02/16633987421/ (CC BY-NC-ND 2.0) vision.communicate

Autovification: Credit: AV Dezign
https://www.flickr.com/photos/91345457@N07/22666878846/ (CC BY-NC-ND 2.0)

Counter image: Dean Hochman Source:
https://www.flickr.com/photos/17997843@N02/24061690099/" (CC BY-NC-ND 2.0)

Cookies: Source Wikipedia
https://en.wikipedia.org/wiki/File:R%C5%AFzn%C3%A9_druhy_cukrov%C3%AD_(2).jpg (CC BY 3.0)



We are Hiring!

Visit our table for more info!

Thanks!

Any questions?

You can find me at:
@mfcabrera
mfcabrera@gmail.com