Coding Bootcamp Code in Python

# THE COLLECTIONS MODULE

# ChainMap

- Provides the ability to link multiple mappings together such that they end up being a single unit

```
from collections import ChainMap

car_parts = {'hood': 500, 'engine': 5000, 'front_door': 750}
car_options = {'A/C': 1000, 'Turbo': 2500, 'rollbar': 300}
car_accessories = {'cover': 100, 'hood_ornament': 150,
'seat_cover': 99}
car_pricing = ChainMap(car_accessories, car_options, car_parts)
print (car_pricing['hood’])

#500
```

# Counter

- Supports convenient and fast tallies
- Also can run it against most iterables

```
from collections import Counter
print (Counter('superfluous'))
#Counter({'u': 3, 's': 2, 'e': 1, 'l': 1, 'f': 1, 'o': 1, 'r': 1,
'p': 1})

counter = Counter('superfluous')
print (counter['u'])
#3
```

# defaultdict

- Subclass of Python's dict that accepts a default_factory as its primary argument

```
from collections import defaultdict

sentence = "The red for jumped over the fence and ran to the zoo
for food"
words = sentence.split(' ')

d = defaultdict(int)
for word in words:
    d[word] += 1

print(d)
```

# deque

- Are a generalization of stacks and queues.

```
from collections import deque
import string

d = deque(string.ascii_lowercase)
for letter in d:
    print(letter)
#a
#b
#c
#...
#z
```

# namedtuple

- Can use to replace Python's tuple

```
from collections import namedtuple

Parts = namedtuple('Parts', 'id_num desc cost amount')
auto_parts = Parts(id_num='1234', desc='Ford Engine',
                   cost=1200.00, amount=10)

print(auto_parts.id_num)

#1234
```

# Code Pack 12

- See the files

1.ChainMap

2.Counter

3.defaultdict

4.deque

5.namedtuple

Coding Bootcamp Code in Python

# ITERATORS AND GENERATORS

# Iterators

- An iterator is an object that will allow you to iterate over a container.

- The iterator in Python is implemented via two distinct methods: __iter__ and __next__.

1. The __iter__ method is required for your container to provide iteration support. It will return the iterator object itself.
2. But if you want to create an iterator object, then you will need to define __next__ as well, which will return the next item in the container.

# Generators

- A normal Python function will always return one value, whether it be a list, an integer or some other object.
- But what if you wanted to be able to call a function and have it yield a series of values? That is where generators come in.
- A generator works by “saving” where it last left off (or **yielding**) and giving the calling function a value. So instead of returning the execution to the caller, it just gives temporary control back.
- a generator function requires Python’s yield statement.

# Code Pack 13

- See the files

1. Iterators
2. Generators

Coding Bootcamp Code in Python

# THE ITERTOOLS MODULE

# The Infinite Iterators

- The itertools package comes with three iterators that can iterate infinitely
- **count(start=0, step=1)**
- **cycle(iterable)**
- **repeat(object)**

# Iterators That Terminate

- Most of the iterators that you create with itertools are not infinite
- **accumulate(iterable)**
- **chain(*iterables)**
- **chain.from_iterable(iterable)**
- **compress(data, selectors)**
- **dropwhile(predicate, iterable)**
- **filterfalse(predicate, iterable)**
- **groupby(iterable, key=None)**
- **islice(iterable, start, stop)**
- **starmap(function, iterable)**
- **takewhile(predicate, iterable)**
- **tee(iterable, n=2)**
- **zip_longest(*iterables, fillvalue=None)**

# The Combinatoric Generators

- The itertools library contains four iterators that can be used for creating combinations and permutations of data.
- **combinations(iterable, r)**
- **combinations_with_replacement(iterable, r)**
- **product(*iterables, repeat=1)**
- **permutations**

# Code Pack 14

- See the files

1.The_Infinite_Iterators

2.Iterators_That_Terminate

3.The_Combinatoric_Generators