DEFAULTDICT

Missing Keys

standard dictionary (`dict`)

```
d = {}
d['a']  → KeyError
```

→ can use the `get` method to handle default values for non-existent keys

`d.get('a', 100)` → we get `100` if `'a'` is not present, but `'a'` is still not in dictionary

`d['a'] = d.get('a', 0) + 1` → general pattern for counters

in general: `d.get(key, value)` → value could be returned from calling a callable

→ works, but possibly have to remember to always use the same default in multiple places for same dict

→ easier would be to define the default once (per dict)

defaultdict

→ `collections` module

→ subclass of `dict` type (`defaultdict` instance IS-A `dict` instance)

→ so has all the functionality of a standard `dict`

```
defaultdict(callable, […])
```

callable is called to calculate a default

remaining arguments are simply passed to `dict` constructor

→ callable must have zero arguments

→ referred to as a factory method

→ default is `None` and `None` will be the default value

```
d = defaultdict(lambda: 'python')
```

d → {}

```
d['a']
```

→ 'python'

d → {'a': 'python'}

→ entry has been created

Other Factory Functions

Often we want to initialize values to 0, an empty string, an empty list, etc

```
int() → 0
```

```
defaultdict(lambda: 0)
defaultdict(int)
```

} has the same effect

```
list() → []
```

```
defaultdict(lambda: [])
defaultdict(list)
```

} has the same effect

→ factory must simply be a callable that can take zero arguments and returns the desired default value

→ can even be a function that calls a database and returns some value

→ factory is invoked every time a default value is needed

→ function does not have to be deterministic

→ can return different values every time it is called

Code