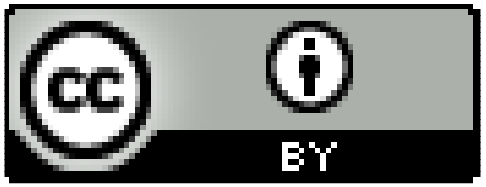
Python

Slicing



Copyright © Software Carpentry 2010
This work is licensed under the Creative Commons Attribution License
See http://software-carpentry.org/license.html for more information.

Lists, strings, and tuples are all *sequences*

Lists, strings, and tuples are all *sequences*

Can be indexed by integers in the range `0…len(X)-1`

Lists, strings, and tuples are all *sequences*

Can be indexed by integers in the range `0…len(X)-1`

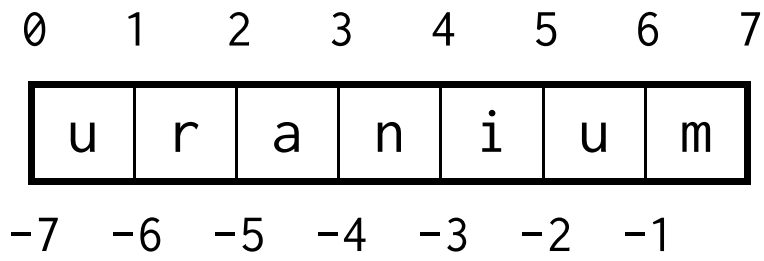
Can also be *sliced* using a range of indices

Lists, strings, and tuples are all *sequences*

Can be indexed by integers in the range `0…len(X)-1`

Can also be *sliced* using a range of indices

```
>>> element = 'uranium'
>>>
```

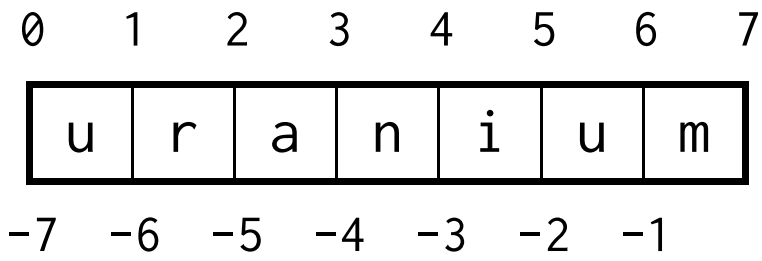


Lists, strings, and tuples are all *sequences*

Can be indexed by integers in the range `0…len(X)-1`

Can also be *sliced* using a range of indices

```
>>> element = 'uranium'
>>> print element[1:4]
ran
>>>
```

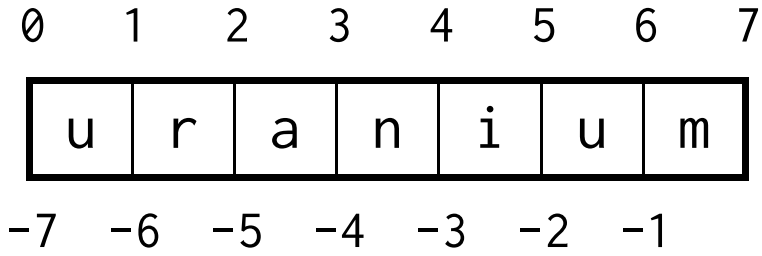


Lists, strings, and tuples are all *sequences*

Can be indexed by integers in the range `0…len(X)-1`

Can also be *sliced* using a range of indices

```
>>> element = 'uranium'
>>> print element[1:4]
ran
>>> print element[:4]
uran
>>>
```

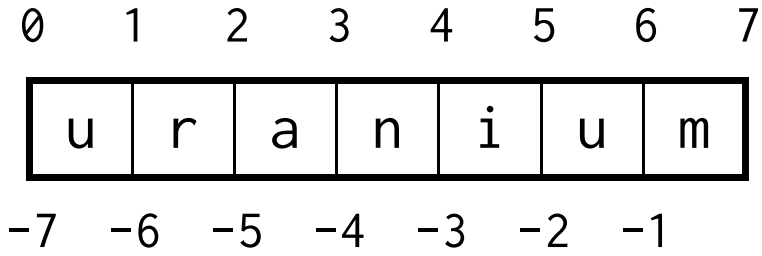


Lists, strings, and tuples are all *sequences*

Can be indexed by integers in the range `0…len(X)-1`

Can also be *sliced* using a range of indices

```
>>> element = 'uranium'
>>> print element[1:4]
ran
>>> print element[:4]
uran
>>> print element[4:]
ium
>>>
```

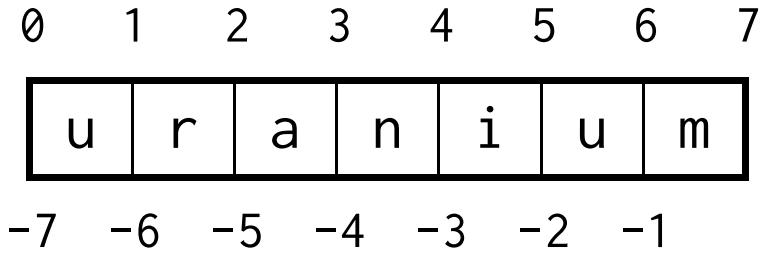


Lists, strings, and tuples are all *sequences*

Can be indexed by integers in the range `0…len(X)-1`

Can also be *sliced* using a range of indices

```
>>> element = 'uranium'
>>> print element[1:4]
ran
>>> print element[:4]
uran
>>> print element[4:]
ium
>>> print element[-4:]
nium
>>>
```



Python checks bounds when indexing

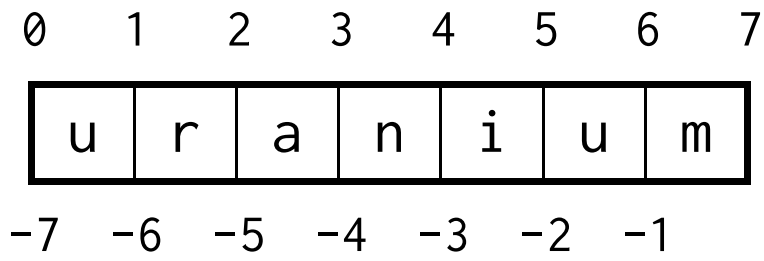
Python checks bounds when indexing

But truncates when slicing

Python checks bounds when indexing

But truncates when slicing

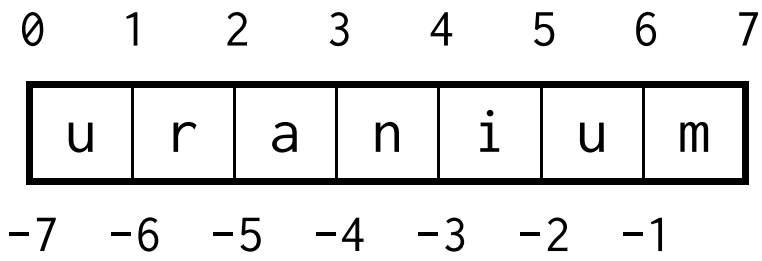
```
>>> element = 'uranium'
>>>
```



Python checks bounds when indexing

But truncates when slicing

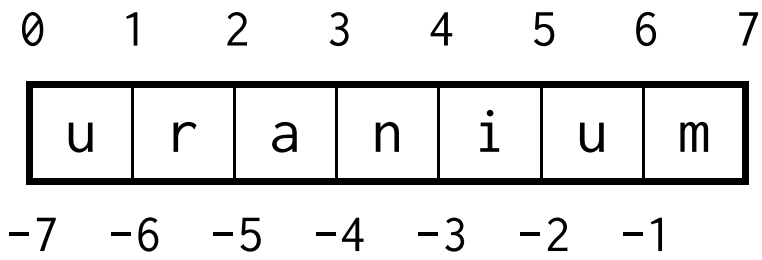
```
>>> element = 'uranium'
>>> print element[400]
IndexError: string index out of range
>>>
```



Python checks bounds when indexing

But truncates when slicing

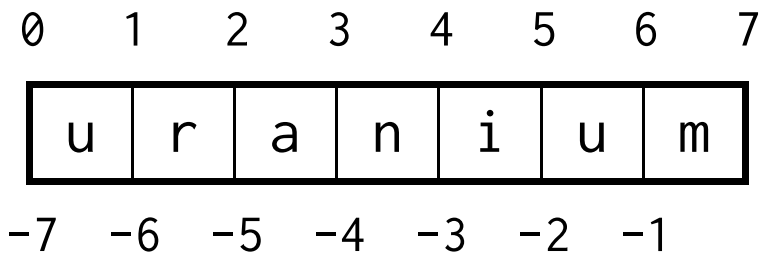
```
>>> element = 'uranium'
>>> print element[400]
IndexError: string index out of range
>>> print element[1:400]
ranium
>>>
```



Python checks bounds when indexing

But truncates when slicing

```
>>> element = 'uranium'
>>> print element[400]
IndexError: string index out of range
>>> print element[1:400]
ranium
>>>
```



"A foolish consistency is

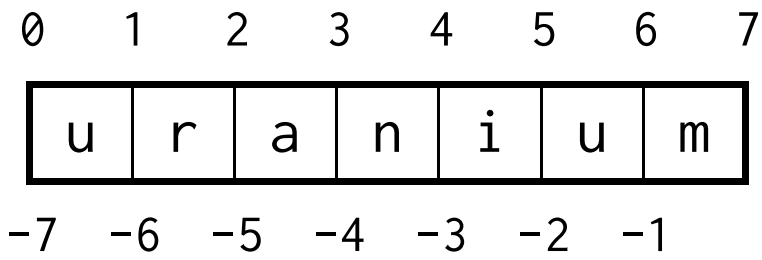
the hobgoblin of little minds."

— *Ralph Waldo Emerson*

Python checks bounds when indexing

But truncates when slicing

```
>>> element = 'uranium'
>>> print element[400]
IndexError: string index out of range
>>> print element[1:400]
ranium
>>>
```



"A foolish consistency is the hobgoblin of little minds."

— *Ralph Waldo Emerson*

"Aw, you're kidding me!"

— *programmers*

So `text[1:3]` is 0, 1, or 2 characters long

So `text[1:3]` is 0, 1, or 2 characters long

`''`	`''`
`'a'`	`''`
`'ab'`	`'b'`
`'abc'`	`'bc'`
`'abcdef'`	`'bc'`

For consistency, `text[1:1]` is the empty string

For consistency, `text[1:1]` is the empty string

- From index 1 up to (but not including) index 1

For consistency, `text[1:1]` is the empty string

- From index 1 up to (but not including) index 1

And `text[3:1]` is always the empty string

For consistency, `text[1:1]` is the empty string

- From index 1 up to (but not including) index 1

And `text[3:1]` is always the empty string

- *Not* the reverse of `text[1:3]`

For consistency, `text[1:1]` is the empty string

- From index 1 up to (but not including) index 1

And `text[3:1]` is always the empty string

- *Not* the reverse of `text[1:3]`

But `text[1:-1]` is everything except the first and last characters

Slicing always creates a new collection

Slicing always creates a new collection

Beware of aliasing

Slicing always creates a new collection

Beware of aliasing

```
>>> points = [[10, 10], [20, 20], [30, 30], [40, 40]]
>>>
```

Slicing always creates a new collection

Beware of aliasing

```
>>> points = [[10, 10], [20, 20], [30, 30], [40, 40]]
>>> middle = points[1:-1]
>>>
```

Slicing always creates a new collection

Beware of aliasing

```
>>> points = [[10, 10], [20, 20], [30, 30], [40, 40]]
>>> middle = points[1:-1]
>>> middle[0][0] = 'whoops'
>>>
```

Slicing always creates a new collection

Beware of aliasing

```
>>> points = [[10, 10], [20, 20], [30, 30], [40, 40]]
>>> middle = points[1:-1]
>>> middle[0][0] = 'whoops'
>>> middle[1][0] = 'aliasing'
>>>
```

Slicing always creates a new collection

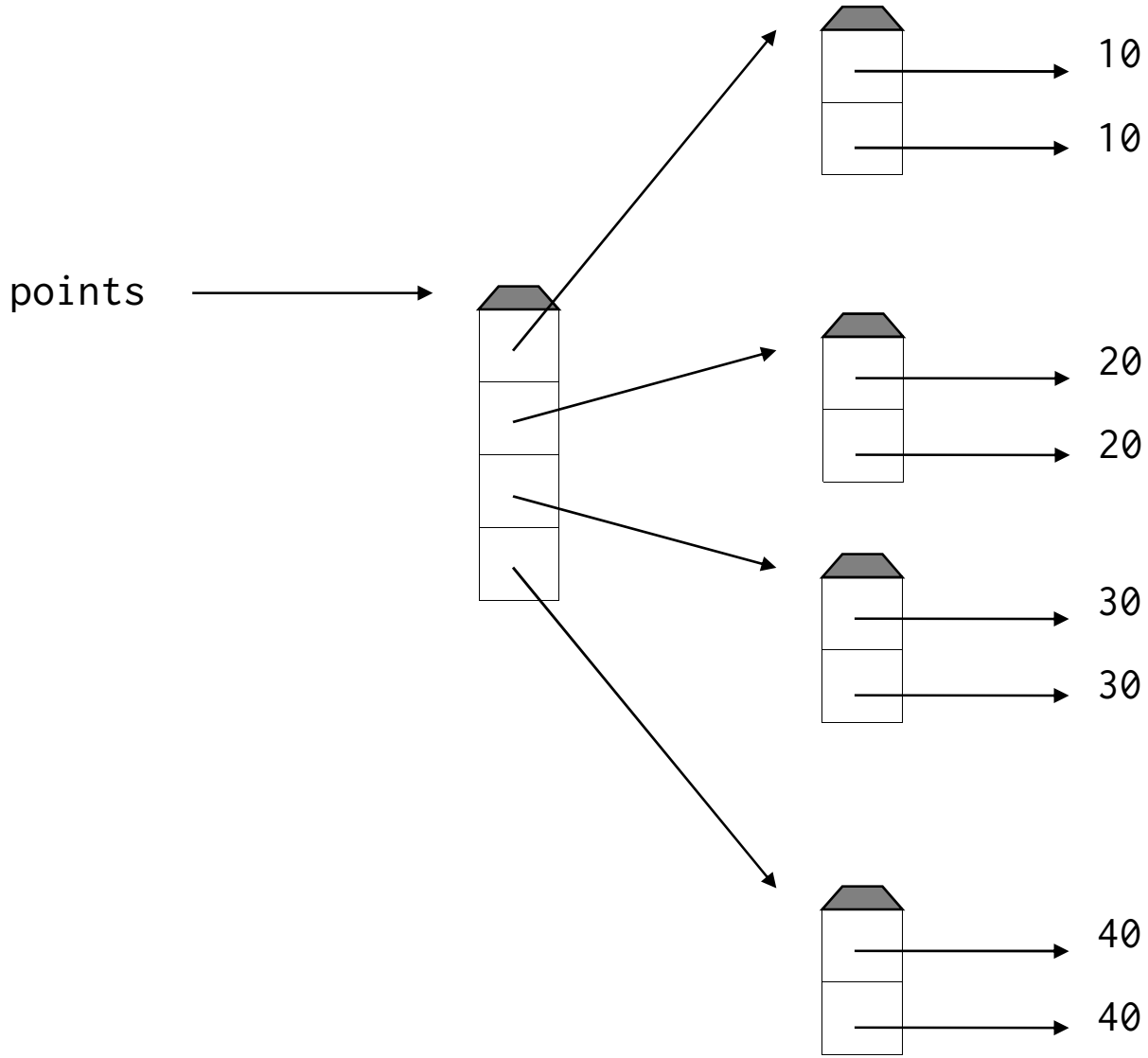
Beware of aliasing

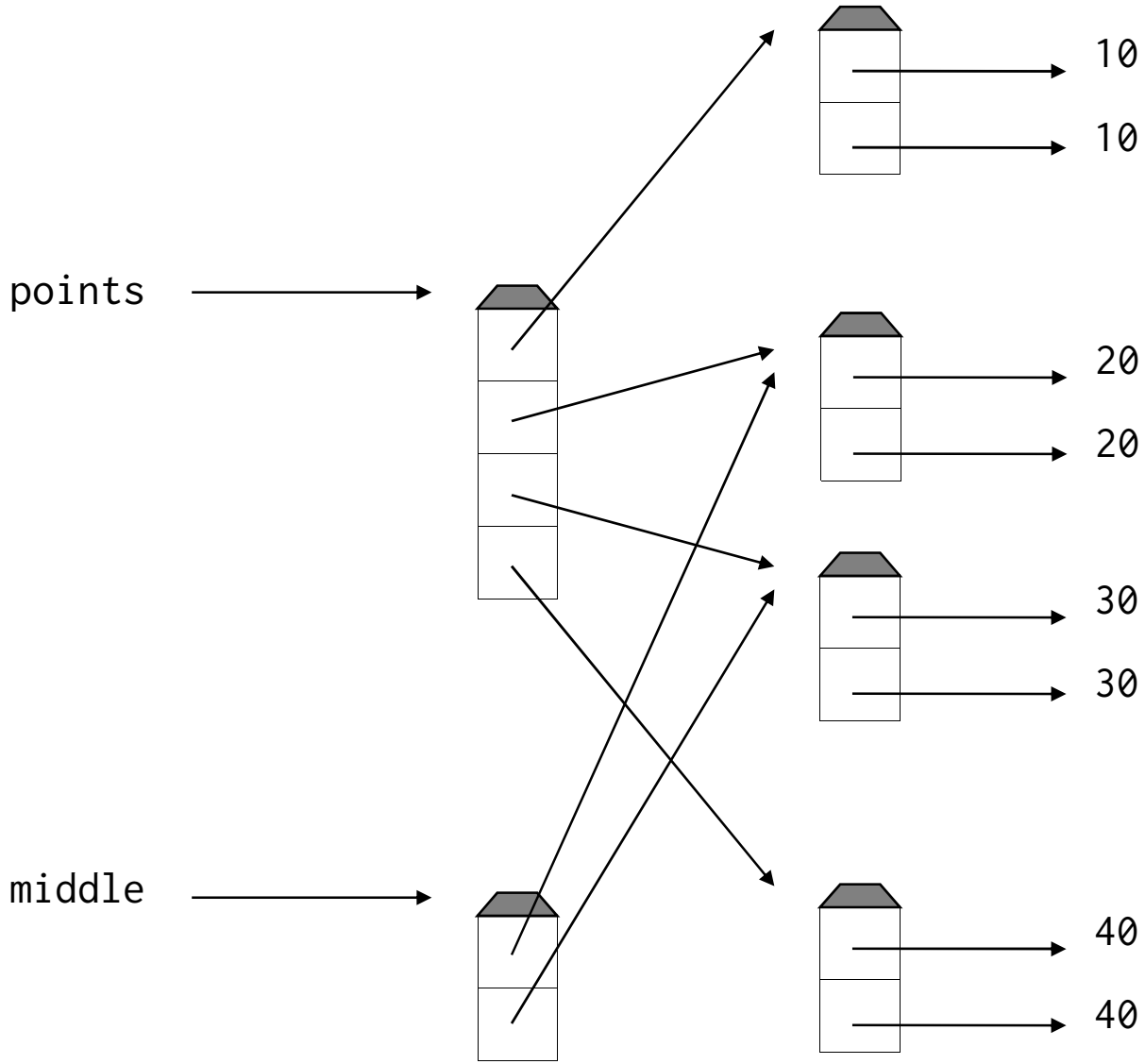
```
>>> points = [[10, 10], [20, 20], [30, 30], [40, 40]]
>>> middle = points[1:-1]
>>> middle[0][0] = 'whoops'
>>> middle[1][0] = 'aliasing'
>>> print middle
[['whoops', 20], ['aliasing', 30]]
>>>
```

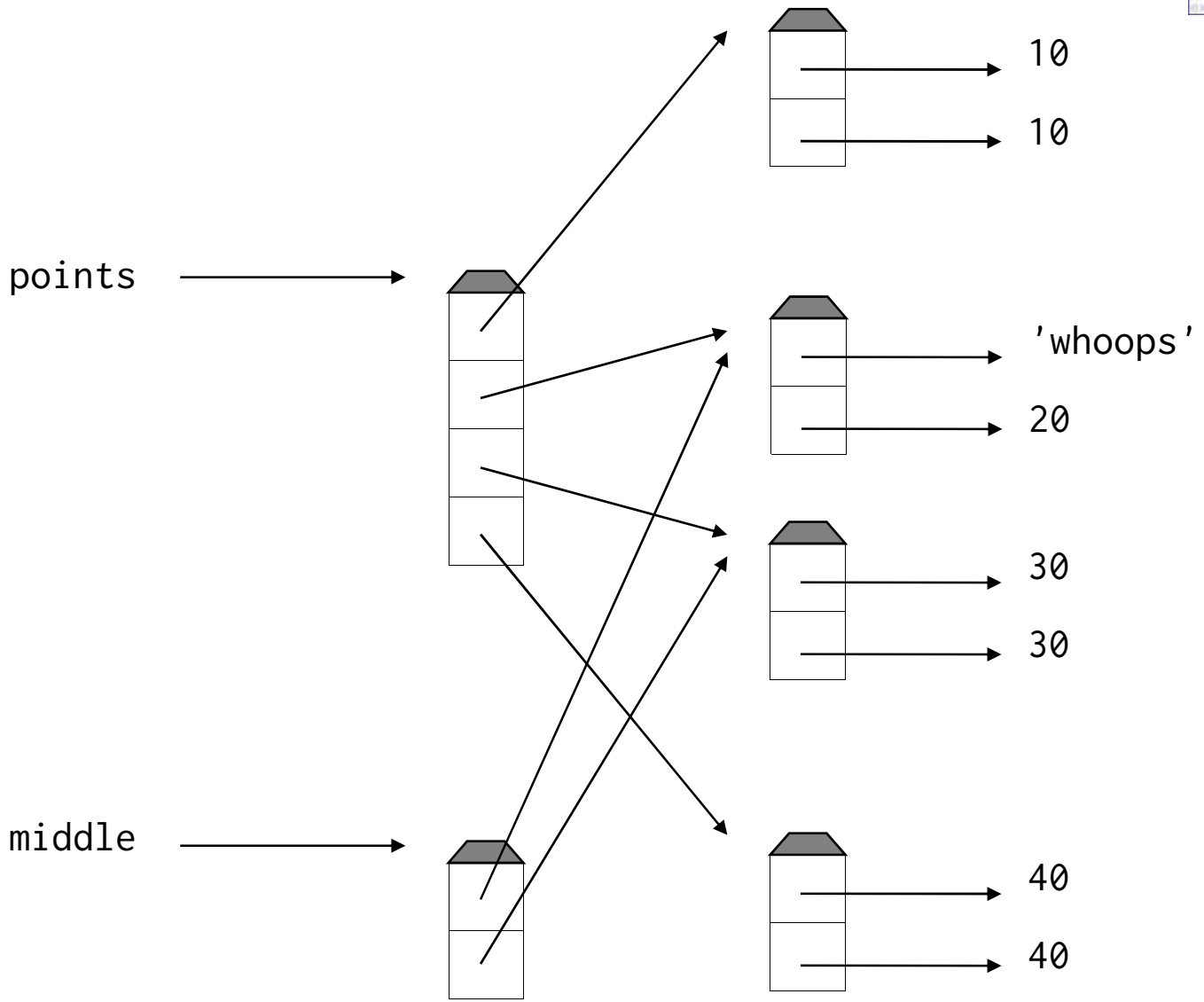
Slicing always creates a new collection

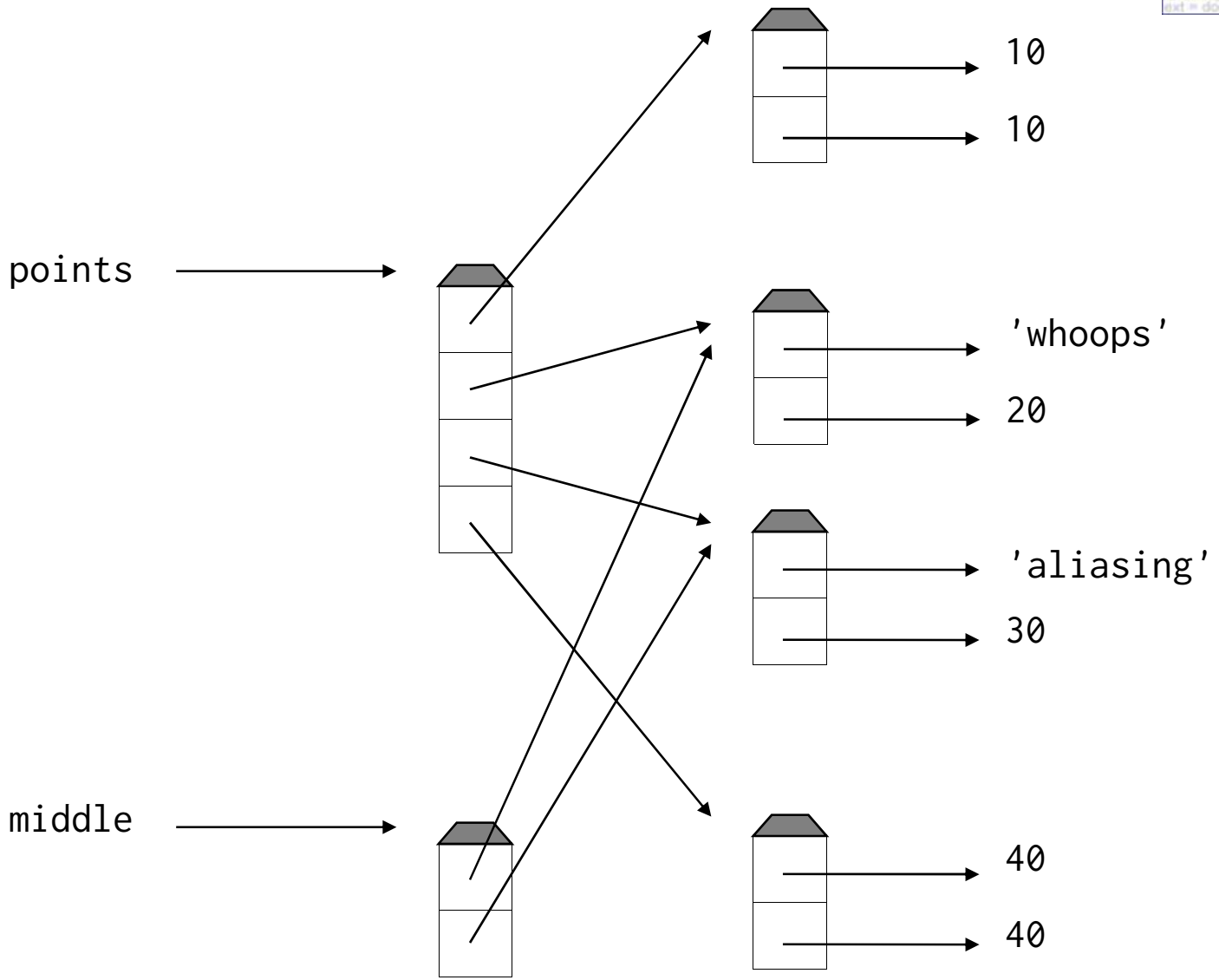
Beware of aliasing

```
>>> points = [[10, 10], [20, 20], [30, 30], [40, 40]]
>>> middle = points[1:-1]
>>> middle[0][0] = 'whoops'
>>> middle[1][0] = 'aliasing'
>>> print middle
[['whoops', 20], ['aliasing', 30]]
>>> print points
[[10, 10], ['whoops', 20], ['aliasing', 30], [40, 40]]
>>>
```


points
10
10
20
20
30
30
40
40


points
middle
10
10
20
20
30
30
40
40

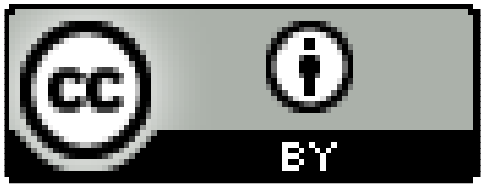

points
middle
10
10
'whoops'
20
30
30
40
40


points
middle
10
10
'whoops'
20
'aliasing'
30
40
40

created by

Greg Wilson

October 2010



Copyright © Software Carpentry 2010
This work is licensed under the Creative Commons Attribution License
See http://software-carpentry.org/license.html for more information.