Python

Slicing

Lists, strings, and tuples are all *sequences*

Lists, strings, and tuples are all *sequences*

Can be indexed by integers in the range $0 \ldots \texttt{len(X)} - 1$

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'
>>>
```

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+
| u | r | a | n | i | u | m |
+---+---+---+---+---+---+---+
 -7  -6  -5  -4  -3  -2  -1
```

Lists, strings, and tuples are all *sequences*
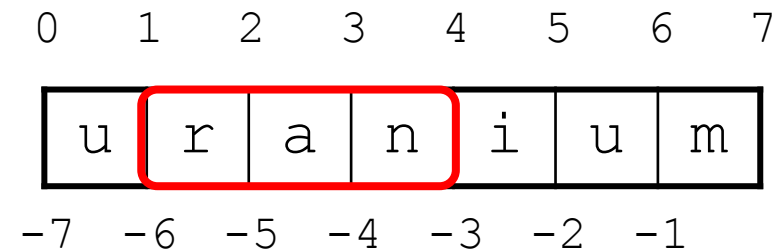
Can be indexed by integers in the range `0…len(X)-1`

Can also be *sliced* using a range of indices

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

Lists, strings, and tuples are all *sequences*

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

Can also be *sliced* using a range of indices

```
>>> element = 'uranium'
>>> print(element[1:4])
ran
>>> print(element[:4])
uran
>>>
```
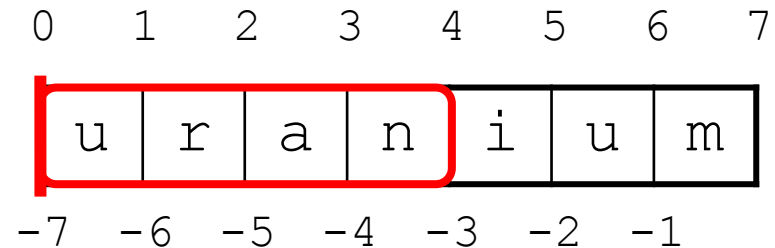
```
0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+
| u | r | a | n | i | u | m |
+---+---+---+---+---+---+---+
-7  -6  -5  -4  -3  -2  -1
```

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

```
>>> element = 'uranium'
>>> print(element[1:4])
ran
>>> print(element[:4])
uran
>>> print(element[4:])
ium
>>>
```
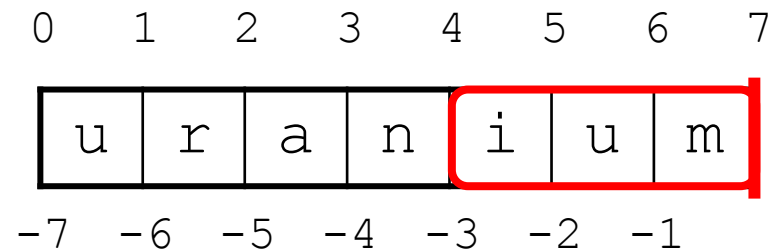
```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+
| u | r | a | n | i | u | m |
+---+---+---+---+---+---+---+
 -7  -6  -5  -4  -3  -2  -1
```

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

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

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+
| u | r | a | n | i | u | m |
+---+---+---+---+---+---+---+
 -7  -6  -5  -4  -3  -2  -1
```

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'
>>>
```

```
0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+
| u | r | a | n | i | u | m |
+---+---+---+---+---+---+---+
  7   6   5   4   3   2   1
```

Python checks bounds when indexing

But truncates when slicing

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

```
  0   1   2   3   4   5   6   7
┌───┬───┬───┬───┬───┬───┬───┐
│ u │ r │ a │ n │ i │ u │ m │
└───┴───┴───┴───┴───┴───┴───┘
  -   -   -   -   -   -   -
  7   6   5   4   3   2   1
```

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
>>>
```

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'
```

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
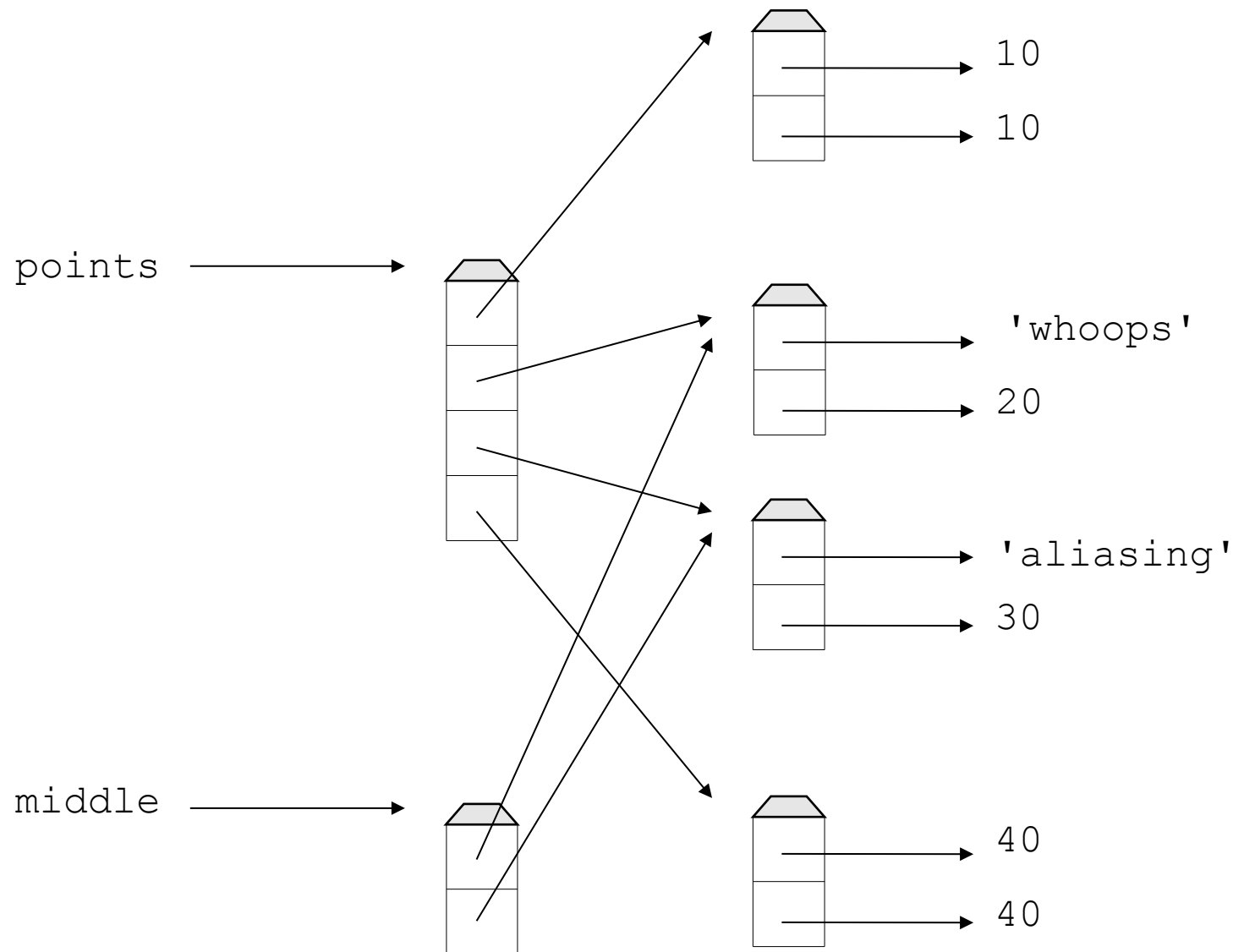
Beware of aliasing

```
>>> 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]]
>>>
```

STOP HERE

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

'aliasing'
30

40
40
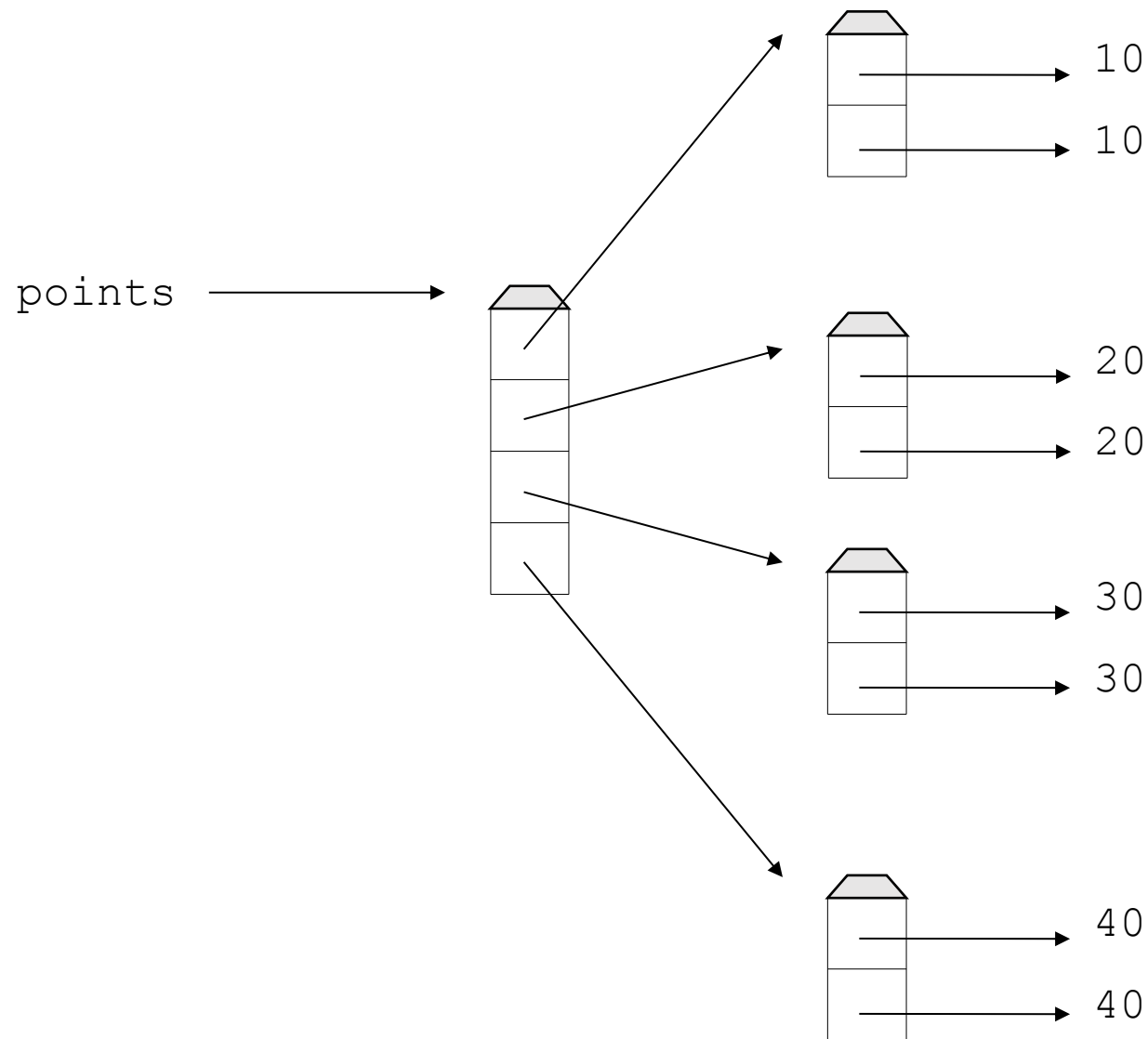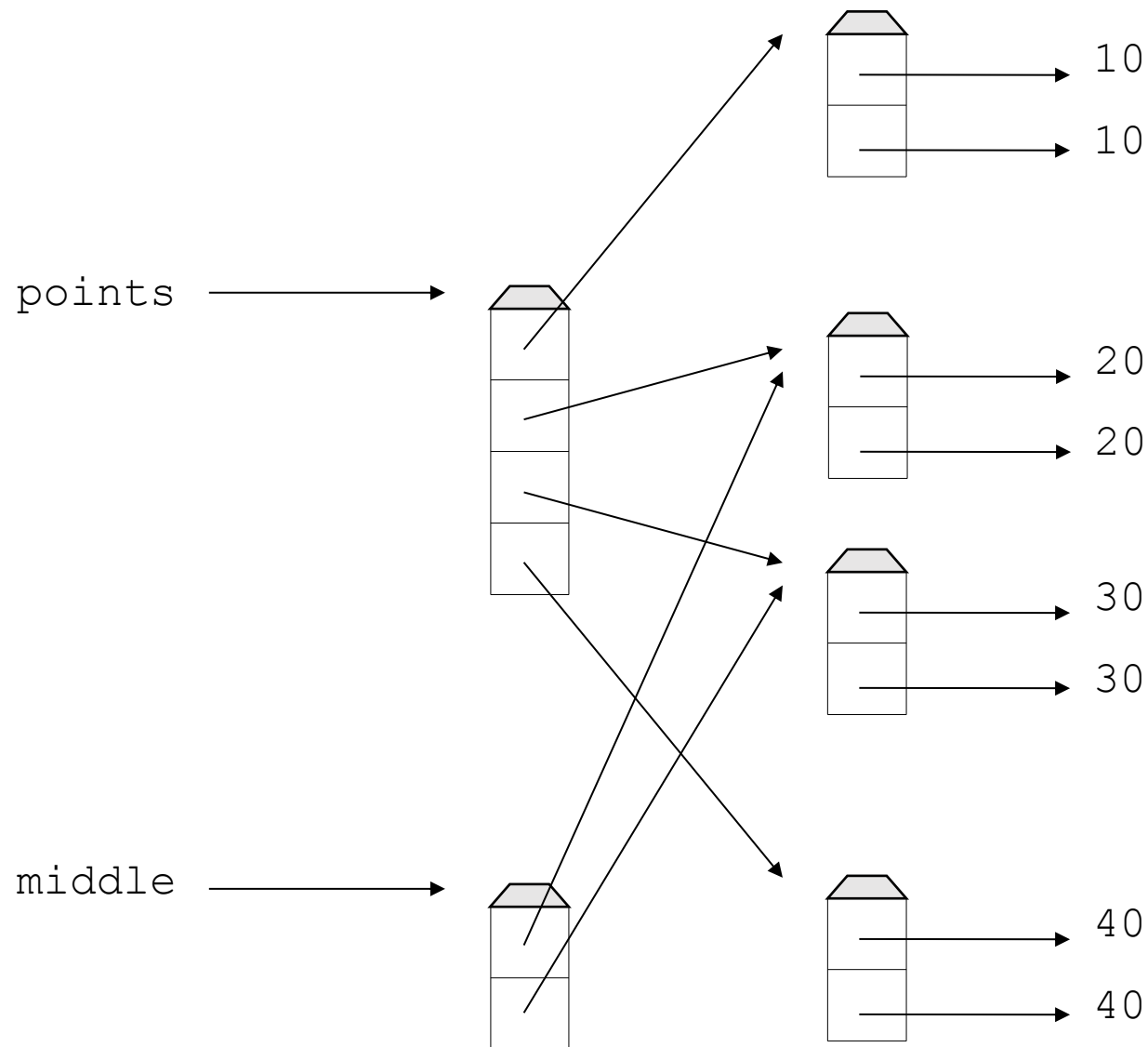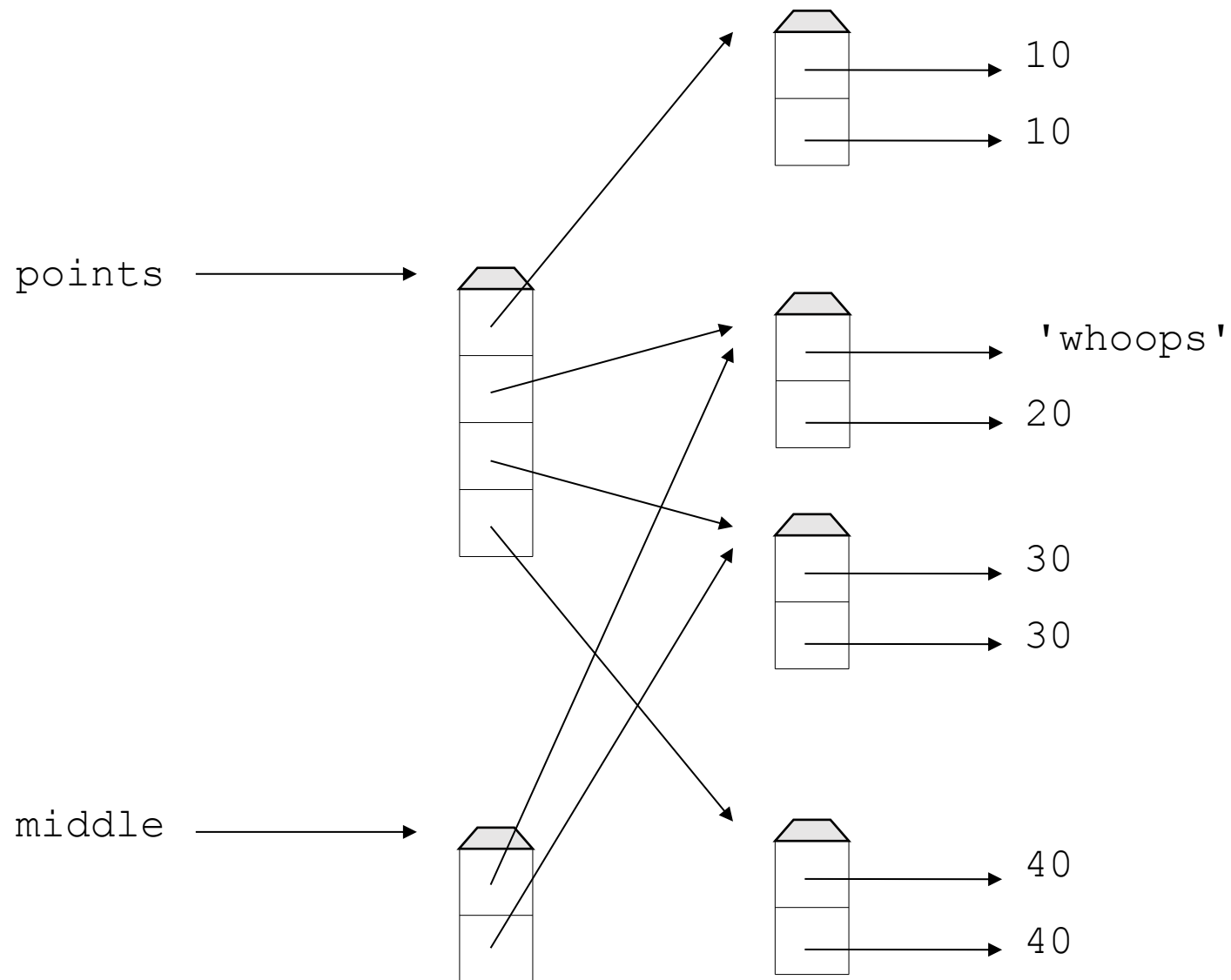
Python

List comprehensions - what are they? They are useful!

List Comprehensions

Python supports a concept called "List Comprehensions". Imagine you want to create a list of square numbers from the list of numbers from 0 to 9. You would type:

```
>>> S = []
>>> for x in range(10):
...        S.append(x**2)


>>> print(S)
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Saving on lines of code

List Comprehensions allow you to do it on **one line**:

```
>>> S = [x**2 for x in range(10)]
>>> print(S)
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

These can be used to construct lists in a natural and easy way.

It gets better - include conditions

Imagine our previous example - but you only want to include values in the list where the result is an even number:

```
>>> S = []
>>> for x in range(10):
...     res = x**2
...     if res % 2 == 0:
...         S.append(res)
>>> print(S)
[0, 4, 16, 36, 64]
```

Can be simplified to...

All one line

```
>>> S =
[x**2 for x in range(10) if x**2 % 2 == 0]


>>> print(S)
[0, 4, 16, 36, 64]
```

See more info at:
https://www.python.org/dev/peps/pep-0202/#examples

created by

Greg Wilson

October 2010