Practical Computing for Scientists

Armin Sobhani

CSCI 2000U

UOIT – Fall 2015

UNIVERSITY OF ONTARIO
INSTITUTE OF TECHNOLOGY

Python
Slicing

by Greg Wilson

UNIVERSITY OF ONTARIO
INSTITUTE OF TECHNOLOGY

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

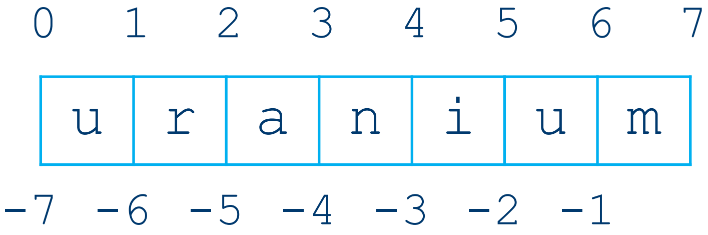
Can also be sliced using a range of indices

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'
>>>
```



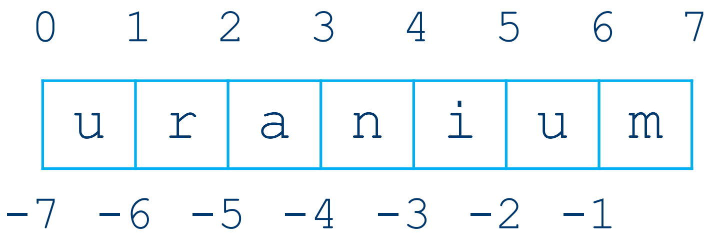
```
 0   1   2   3   4   5   6   7
   u   r   a   n   i   u   m
-7  -6  -5  -4  -3  -2  -1
```

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
>>>
```



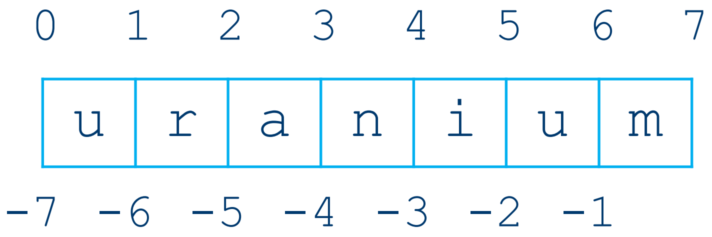
```
 0   1   2   3   4   5   6   7
   u   r   a   n   i   u   m
-7  -6  -5  -4  -3  -2  -1
```

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
>>>
```



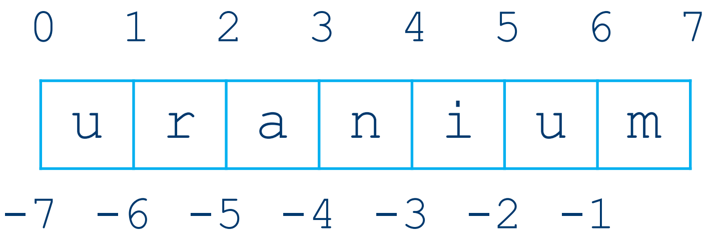
```
 0   1   2   3   4   5   6   7
   u   r   a   n   i   u   m
-7  -6  -5  -4  -3  -2  -1
```

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
>>>
```



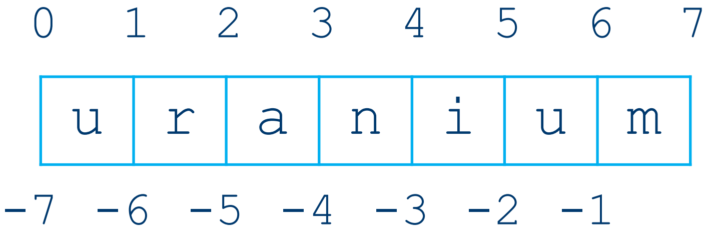
```
 0   1   2   3   4   5   6   7
 | u | r | a | n | i | u | m |
-7  -6  -5  -4  -3  -2  -1
```

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

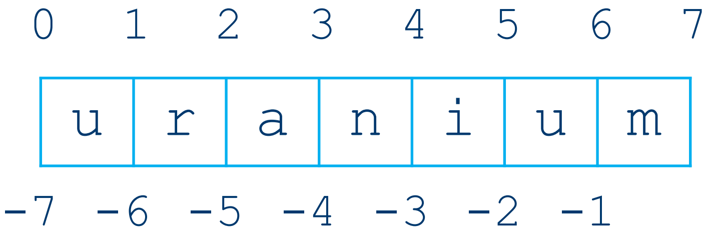
Python checks bounds when indexing

But truncates when slicing

Python checks bounds when indexing

But truncates when slicing

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

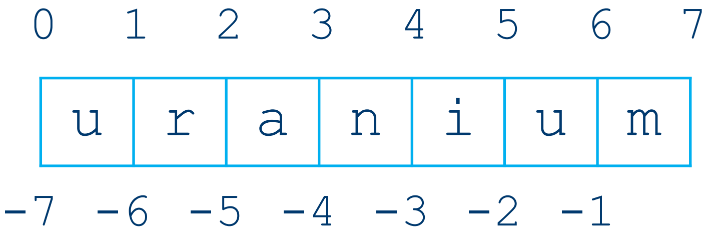


```
 0   1   2   3   4   5   6   7
   u   r   a   n   i   u   m
-7  -6  -5  -4  -3  -2  -1
```

Python checks bounds when indexing

But truncates when slicing

```
>>> element = 'uranium'
>>> print(element[400])
```



0 1 2 3 4 5 6 7

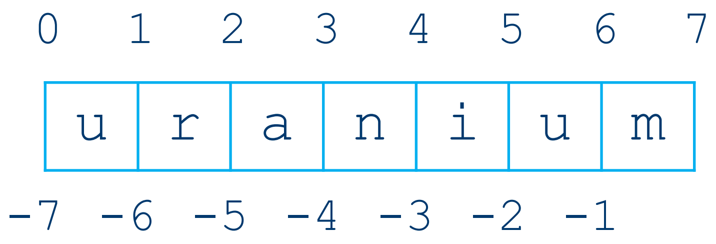
u r a n i u m

-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
>>>
```

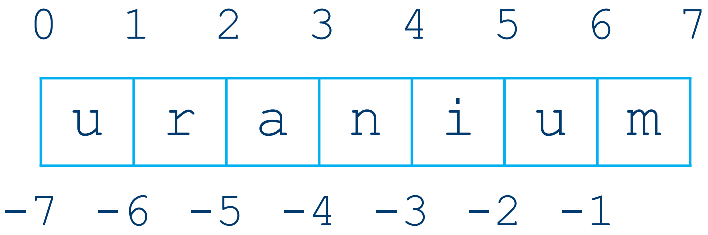


```
 0    1    2    3    4    5    6    7
   u    r    a    n    i    u    m
-7   -6   -5   -4   -3   -2   -1
```

Python checks bounds when indexing

But truncates when slicing

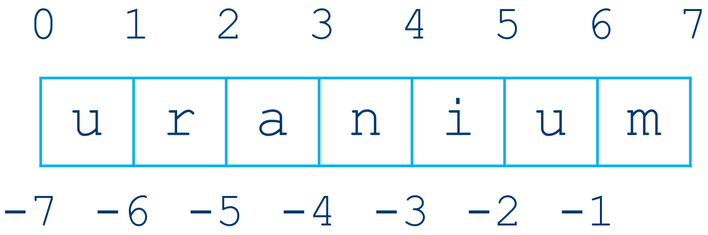
```
>>> element = 'uranium'
>>> print(element[400])
IndexError: string index out of range
>>> print(element[1:400])
>>> ranium
>>>
```



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
>>>
```



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

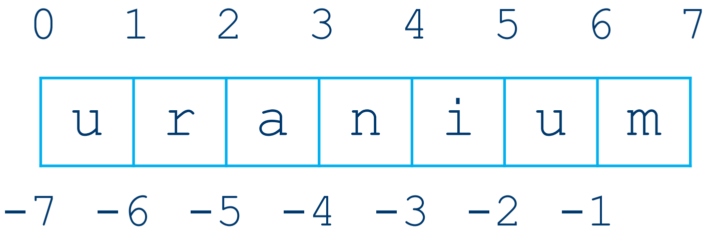
"A foolish consistency is
the hobgoblin of little minds."

— *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
>>>
```



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

"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

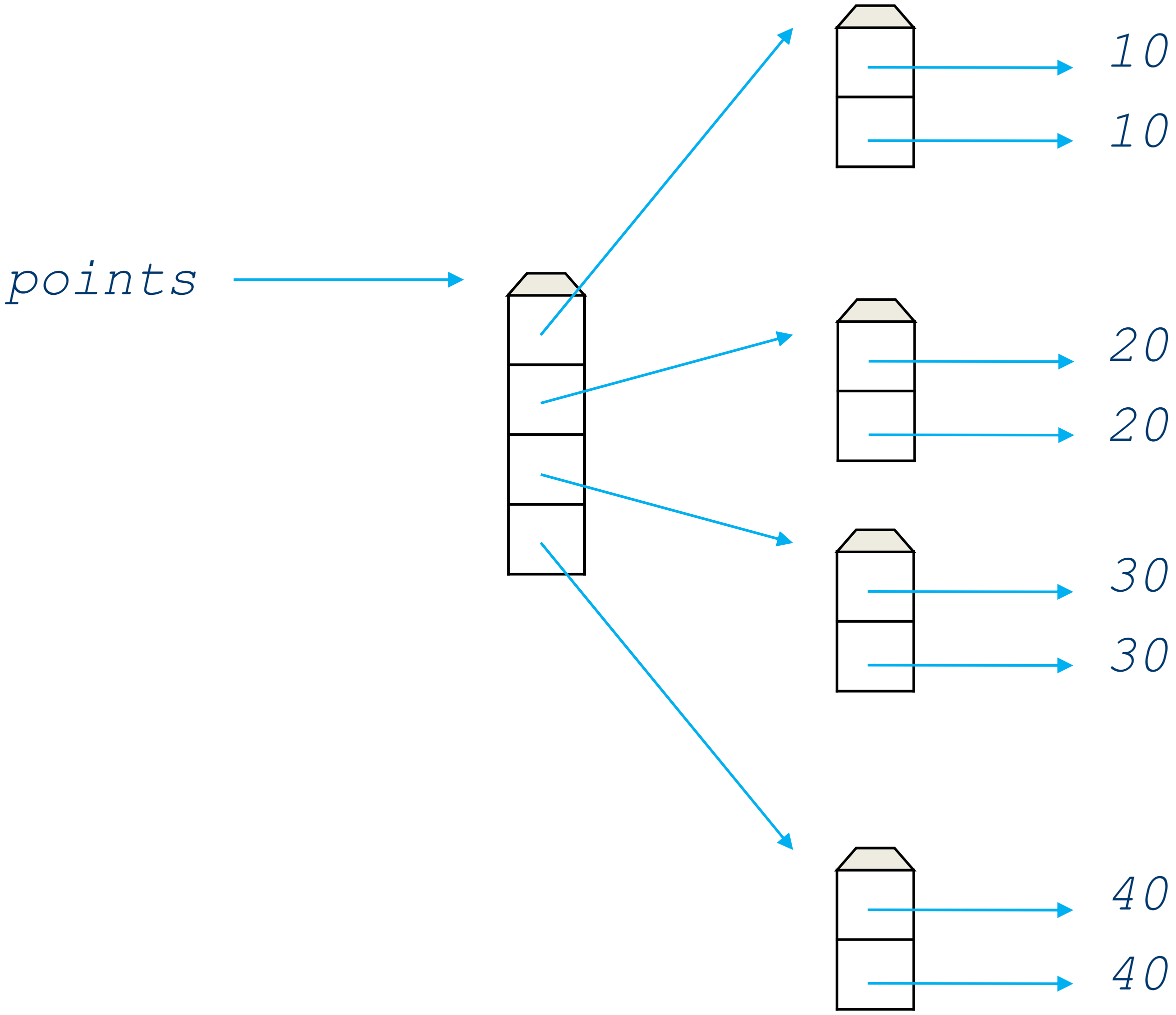
Beware of aliasing

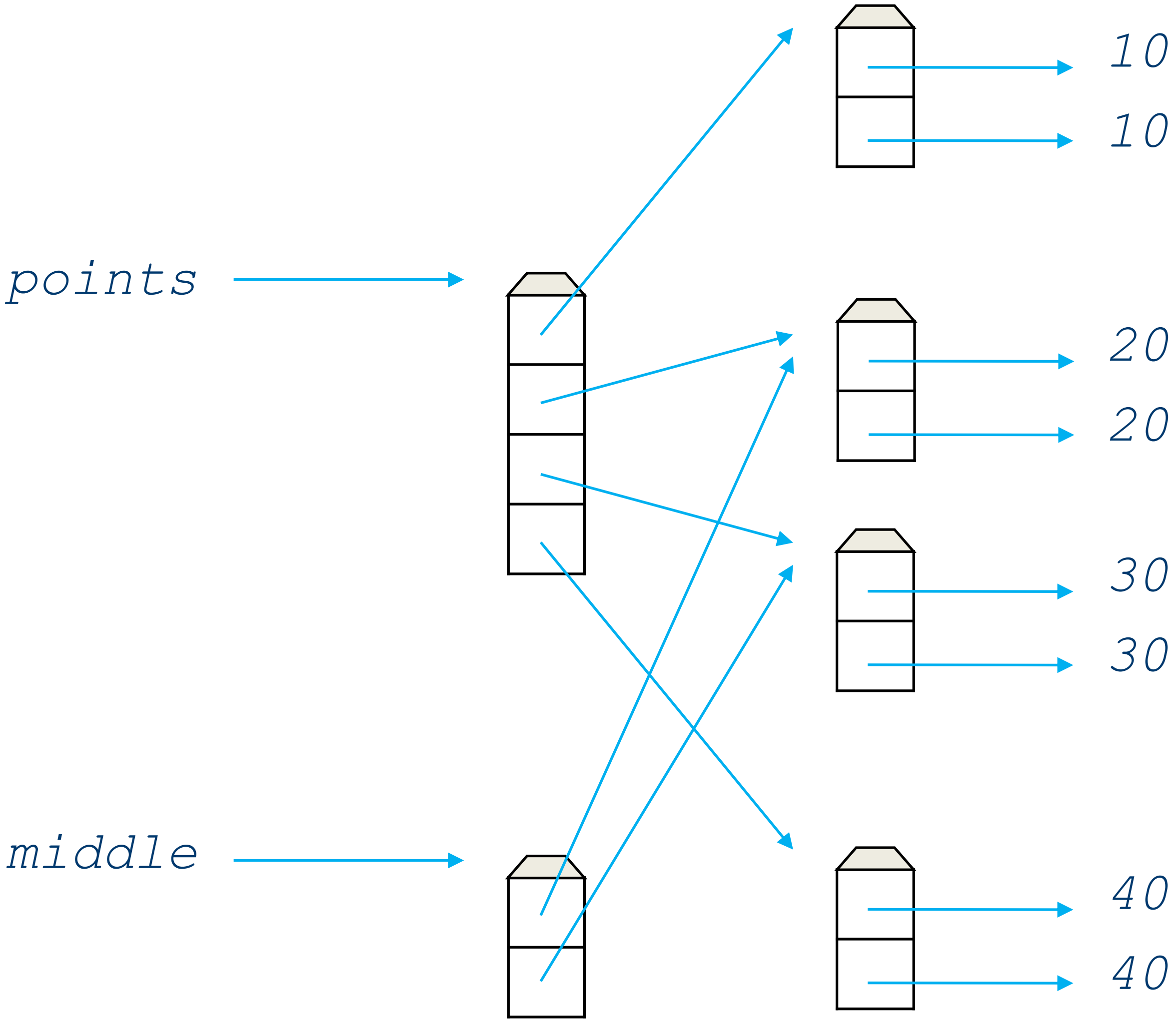
```
>>> points = [[10, 10], [20, 20], [30, 30], [40, 40]]
>>> middle = points[1:-1]
>>> middle[0][0] = 'whoops'
>>> middle[1][0] = 'aliasing'
>>> print(middle)
>>> [['whoops', 20], ['aliasing', 30]]
>>>
```

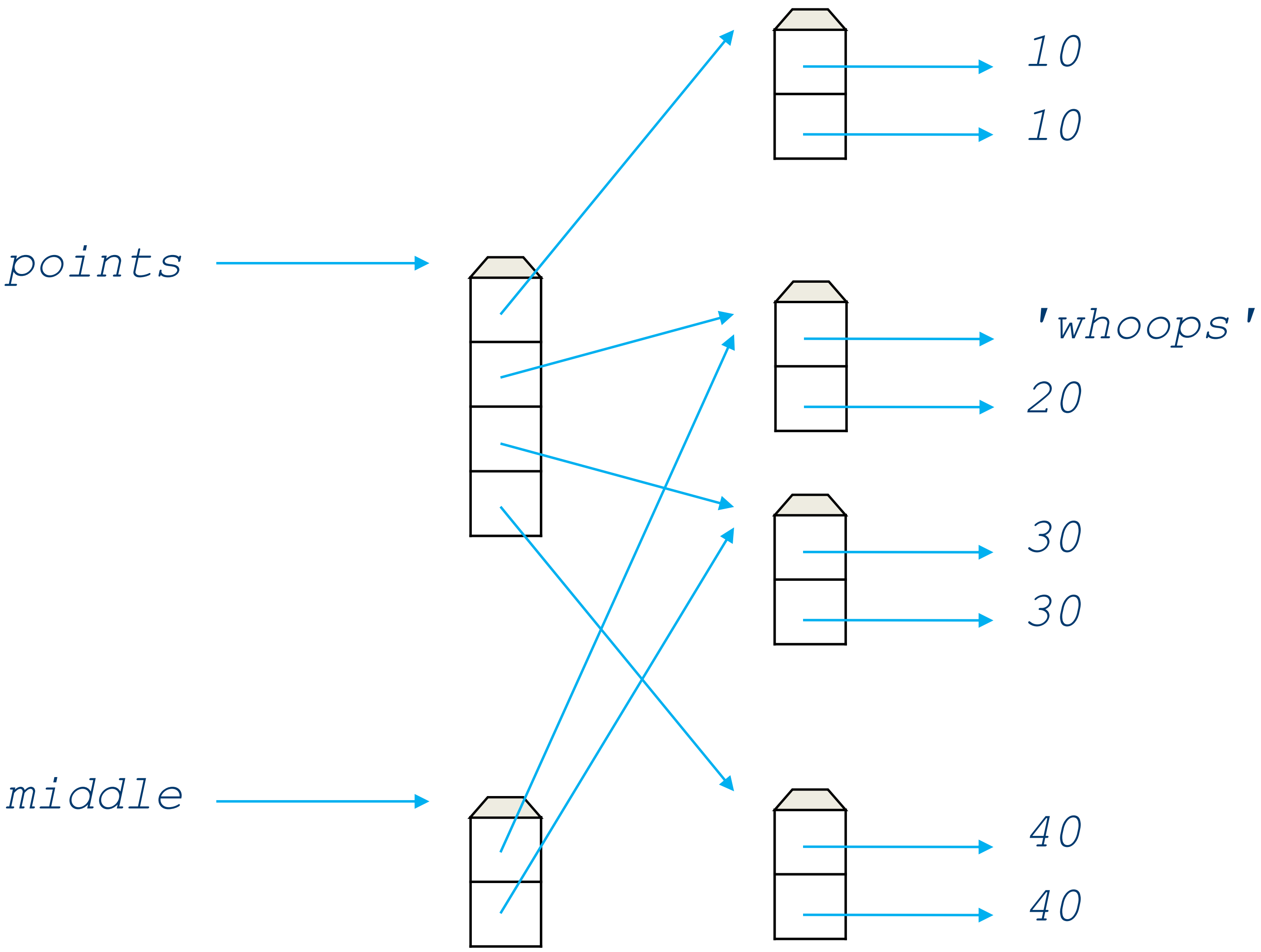
Slicing always creates a new collection

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'

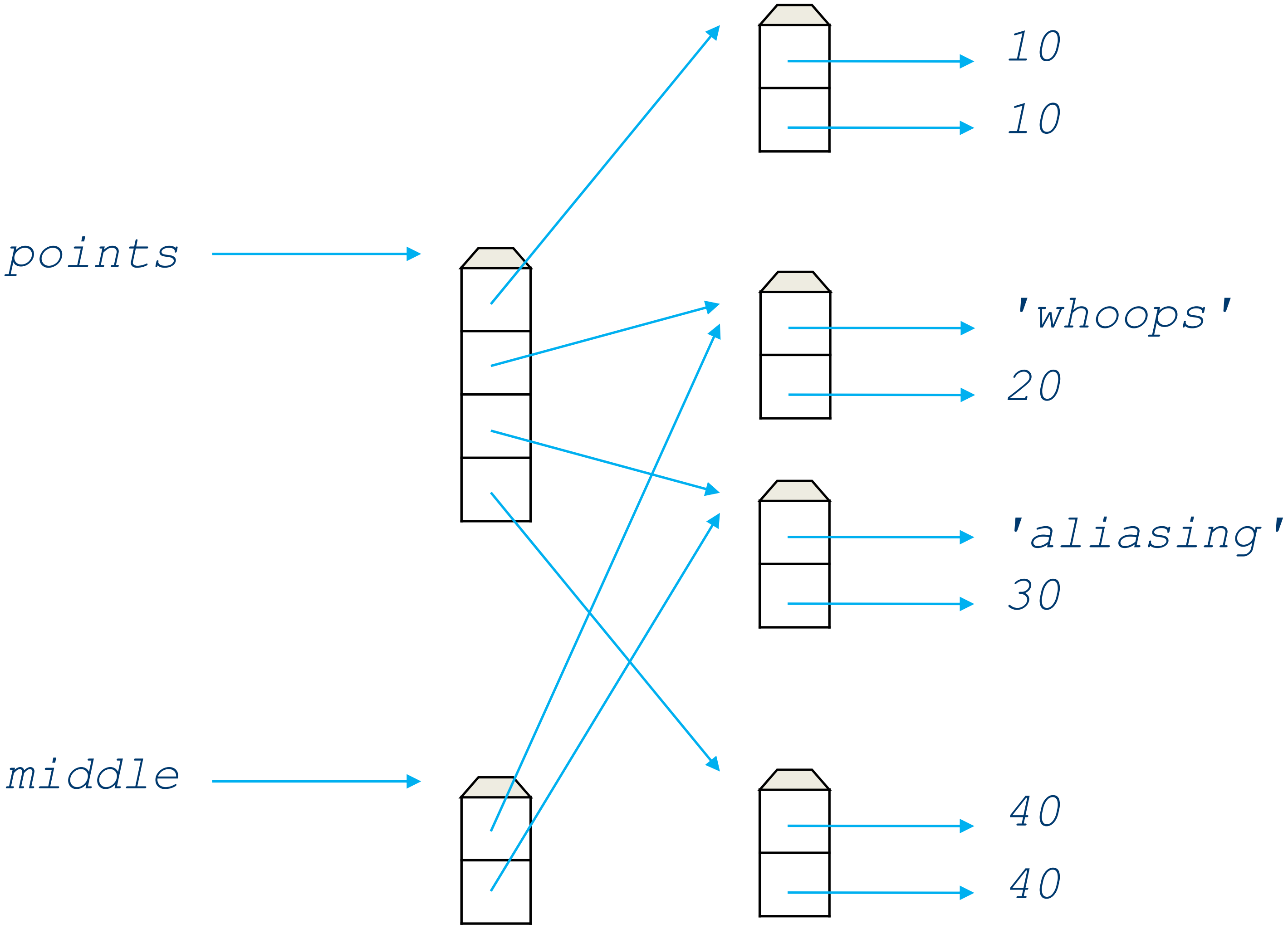
20

30

30

40

40



points

middle

10

10

'whoops'

20

'aliasing'

30

40

40

Python

NumPy

UNIVERSITY OF ONTARIO INSTITUTE OF TECHNOLOGY

NumPy

NumPy

the fundamental package for scientific computing with Python:

NumPy

the fundamental package for scientific computing with Python:

- N-dimensional array object

NumPy

the fundamental package for scientific computing with Python:

- N-dimensional array object
- linear algebra

NumPy

the fundamental package for scientific computing with Python:

- N-dimensional array object
- linear algebra
- Fourier transform

NumPy

the fundamental package for scientific computing with Python:

- N-dimensional array object
- linear algebra
- Fourier transform
- random number capabilities

Importing the NumPy Module

```
>>> import numpy
```

Importing the NumPy Module

```
>>> import numpy
>>> import numpy as np
```

Importing the NumPy Module

```
>>> import numpy
>>> import numpy as np
>>> from numpy import *
```

Importing the NumPy Module

```
>>> import numpy
>>> import numpy as np
~~>>> from numpy import *~~
```

Importing the NumPy Module

```
>>> import numpy
>>> import numpy as np
```

~~>>> from numpy import *~~

NumPy Arrays

```
>>> a = np.array([1, 4, 5, 8], float)
```

NumPy Arrays

```
>>> a = np.array([1, 4, 5, 8], float)
```

arrays are similar to lists in Python

NumPy Arrays

```
>>> a = np.array([1, 4, 5, 8], float)
```

arrays are similar to lists in Python

except that every element of an array must be of the same type

NumPy Arrays

```
>>> a = np.array([1, 4, 5, 8], float)
```

arrays are similar to lists in Python except that every element of an array must be of the same type

typically a numeric type like **`float`** or **`int`**

NumPy Arrays

```
>>> a = np.array([1, 4, 5, 8], float)
>>> a
```

NumPy Arrays

```
>>> a = np.array([1, 4, 5, 8], float)
>>> a
array([ 1., 4., 5., 8.])
```

NumPy Arrays

```
>>> a = np.array([1, 4, 5, 8], float)
>>> a
array([ 1., 4., 5., 8.])
>>> type(a)
```

NumPy Arrays

```
>>> a = np.array([1, 4, 5, 8], float)
>>> a
array([ 1.,  4.,  5.,  8.])
>>> type(a)
<type 'numpy.ndarray'>
```

Accessed and Sliced Like Lists

```
>>> a = np.array([1, 4, 5, 8], float)
```

Accessed and Sliced Like Lists

```
>>> a = np.array([1, 4, 5, 8], float)
>>> a[:2]
```

Accessed and Sliced Like Lists

```
>>> a = np.array([1, 4, 5, 8], float)
>>> a[:2]
array([ 1., 4.])
```

Accessed and Sliced Like Lists

```
>>> a = np.array([1, 4, 5, 8], float)
>>> a[:2]
array([ 1., 4.])
>>> a[3]
```

Accessed and Sliced Like Lists

```
>>> a = np.array([1, 4, 5, 8], float)
>>> a[:2]
array([ 1., 4.])
>>> a[3]
8.0
```

Accessed and Sliced Like Lists

```
>>> a = np.array([1, 4, 5, 8], float)
>>> a[:2]
array([ 1.,  4.])
>>> a[3]
8.0
>>> a[0] = 5.
```

Accessed and Sliced Like Lists

```
>>> a = np.array([1, 4, 5, 8], float)
>>> a[:2]
array([ 1., 4.])
>>> a[3]
8.0
>>> a[0] = 5.
>>> a
array([ 5., 4., 5., 8.])
```

Multidimensional Arrays

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]], float)
```

Multidimensional Arrays

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]], float)
>>> a
array([[ 1., 2., 3.],
       [ 4., 5., 6.]])
```

Multidimensional Arrays

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]], float)
>>> a
array([[ 1.,  2.,  3.],
       [ 4.,  5.,  6.]])
>>> a[0,0]
1.0
```

Multidimensional Arrays

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]], float)
>>> a
array([[ 1., 2., 3.],
       [ 4., 5., 6.]])
>>> a[0,0]
1.0
>>> a[0,1]
2.0
```

Slicing of Multidimensional Arrays

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]], float)
```

Slicing of Multidimensional Arrays

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]], float)
>>> a[1,:]
array([ 4., 5., 6.])
```

Slicing of Multidimensional Arrays

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]], float)
>>> a[1,:]
array([ 4., 5., 6.])
```

in a dimension indicates the use of everything along that dimension

Slicing of Multidimensional Arrays

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]], float)
>>> a[1,:]
array([ 4., 5., 6.])
>>> a[:,2]
array([ 3., 6.])
```

Slicing of Multidimensional Arrays

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]], float)
>>> a[1,:]
array([ 4., 5., 6.])
>>> a[:,2]
array([ 3., 6.])
>>> a[-1:,-2:]
array([[ 5., 6.]])
```

Methods of Arrays

```
>>> X = np.zeros((2, 3))
```

Methods of Arrays

```
>>> X = np.zeros((2, 3))
>>> X
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
```

Methods of Arrays

```
>>> X = np.zeros((2, 3))
>>> X
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
>>> a.shape
(2, 3)
```

Methods of Arrays

```
>>> X = np.zeros((2, 3))
>>> X
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
>>> a.shape
(2, 3)
>>> a.dtype
dtype('float64')
```

Random Numbers

```
>>> np.random.seed(293423)
```

Random Numbers

```
>>> np.random.seed(293423)
>>> np.random.rand(5)
array([ 0.33677247,  0.52693437,  0.79529578])
```

Random Numbers

```
>>> np.random.seed(293423)
>>> np.random.rand(5)
array([ 0.33677247,  0.52693437,  0.79529578])
>>> np.random.rand(2,3)
array([[ 0.78867702,  0.02147624,  0.84612516],
       [ 0.0704939 ,  0.1526965 ,  0.77831701]])
```