Science & Technology Facilities Council
UK Research and Innovation
NERC SCIENCE OF THE ENVIRONMENT

Python

Lists

Centre for Environmental Data Analysis
SCIENCE AND TECHNOLOGY FACILITIES COUNCIL
NATURAL ENVIRONMENT RESEARCH COUNCIL

software carpentry

National Centre for Atmospheric Science
NATURAL ENVIRONMENT RESEARCH COUNCIL

National Centre for Earth Observation
NATURAL ENVIRONMENT RESEARCH COUNCIL

Loops let us do things many times

Loops let us do things many times

Collections let us store many values together

Loops let us do things many times

Collections let us store many values together

Most popular collection is a *list*

Create using `[value, value, ...]`

Create using `[value, value, ...]`

Get/set values using `var[index]`

Create using `[value, value, ...]`

Get/set values using `var[index]`

```
gases = ['He', 'Ne', 'Ar', 'Kr']
print(gases)
['He', 'Ne', 'Ar', 'Kr']
```

Create using `[value, value, ...]`

Get/set values using `var[index]`

```
gases = ['He', 'Ne', 'Ar', 'Kr']
print(gases)
['He', 'Ne', 'Ar', 'Kr']

print(gases[1])
Ne
```

Index from 0, not 1

Index from 0, not 1

Reasons made sense for C in 1970...

Index from 0, not 1

Reasons made sense for C in 1970...

It's an error to try to access out of range

Index from 0, not 1

Reasons made sense for C in 1970...

It's an error to try to access out of range

```
gases = ['He', 'Ne', 'Ar', 'Kr']
print(gases[4])
IndexError: list index out of range
```

Use `len(list)` to get length of list

Use `len(list)` to get length of list

```
gases = ['He', 'Ne', 'Ar', 'Kr']
print(len(gases))
4
```

Use `len(list)` to get length of list

```
gases = ['He', 'Ne', 'Ar', 'Kr']
print(len(gases))
4
```

Returns 0 for the *empty list*

```
etheric = []
print(len(etheric))
0
```

Some negative indices work

Some negative indices work

`values[-1]` is last element, `values[-2]` next-to-last, ...

Some negative indices work

`values[-1]` is last element, `values[-2]` next-to-last, ...

```
gases = ['He', 'Ne', 'Ar', 'Kr']
```

Some negative indices work

`values[-1]` is last element, `values[-2]` next-to-last, ...

```
gases = ['He', 'Ne', 'Ar', 'Kr']
print(gases[-1], gases[-4])
Kr He
```

Some negative indices work

`values[-1]` is last element, `values[-2]` next-to-last, ...

```
gases = ['He', 'Ne', 'Ar', 'Kr']
print(gases[-1], gases[-4])
Kr He
```

`values[-1]` is much nicer than `values[len(values)-1]`

Some negative indices work

`values[-1]` is last element, `values[-2]` next-to-last, ...

```
gases = ['He', 'Ne', 'Ar', 'Kr']
print(gases[-1], gases[-4])
Kr He
```

`values[-1]` is much ~~nicer~~ than `values[len(values)-1]`

less error prone

Mutable : can change it after it is created

Mutable : can change it after it is created

```
gases = ['He', 'Ne', 'Ar', 'K']  # last entry misspelled
```

Mutable : can change it after it is created

```
gases = ['He', 'Ne', 'Ar', 'K']  # last entry misspelled
gases[3] = 'Kr'
```

Mutable : can change it after it is created

```
gases = ['He', 'Ne', 'Ar', 'K']  # last entry misspelled
gases[3] = 'Kr'
print(gases)
['He', 'Ne', 'Ar', 'Kr']
```

Mutable : can change it after it is created

```
gases = ['He', 'Ne', 'Ar', 'K']  # last entry misspelled
gases[3] = 'Kr'
print(gases)
['He', 'Ne', 'Ar', 'Kr']
```

Location must exist before assignment

Mutable : can change it after it is created

```
gases = ['He', 'Ne', 'Ar', 'K']  # last entry misspelled
gases[3] = 'Kr'
print(gases)
['He', 'Ne', 'Ar', 'Kr']
```

Location must exist before assignment

```
gases = ['He', 'Ne', 'Ar', 'Kr']
```

Mutable : can change it after it is created

```
gases = ['He', 'Ne', 'Ar', 'K']  # last entry misspelled
gases[3] = 'Kr'
print(gases)
['He', 'Ne', 'Ar', 'Kr']
```

Location must exist before assignment

```
gases = ['He', 'Ne', 'Ar', 'Kr']
gases[4] = 'Xe'
IndexError: list assignment index out of range
```

Heterogeneous : can store values of many kinds

Heterogeneous : can store values of many kinds

```
helium = ['He', 2]
neon = ['Ne', 8]
```

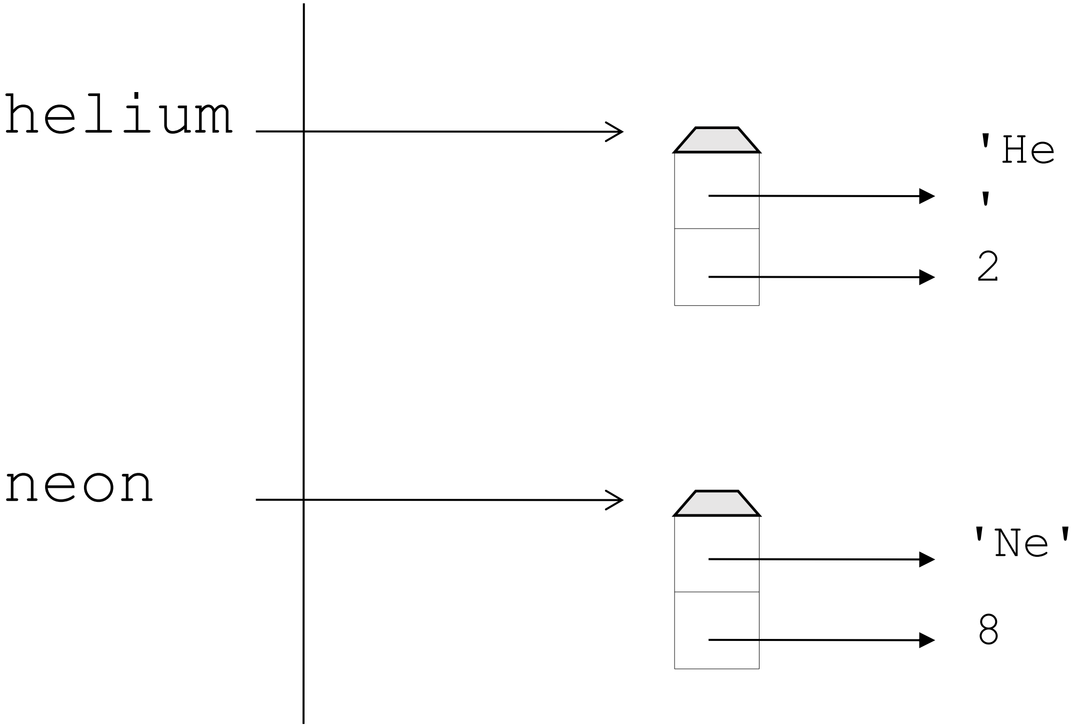
Heterogeneous : can store values of many kinds

```
helium = ['He', 2]
neon = ['Ne', 8]
```

[string, int]

Heterogeneous : can store values of many kinds

```
helium = ['He', 2]
neon = ['Ne', 8]
```

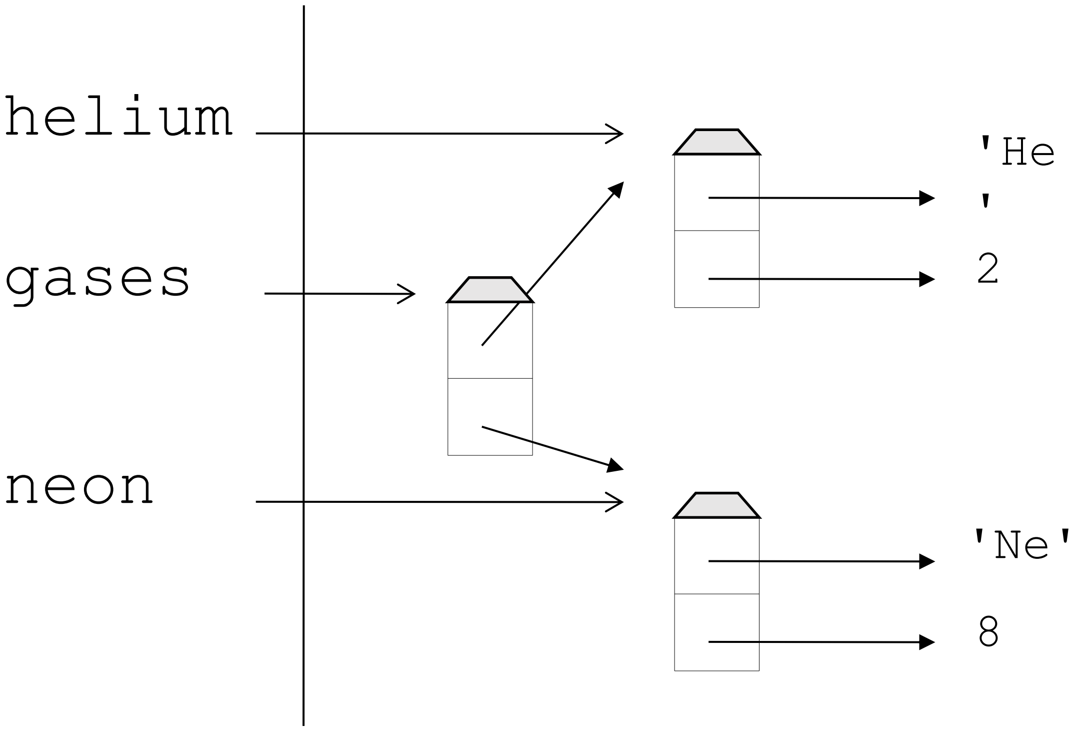


Heterogeneous : can store values of many kinds

```
helium = ['He', 2]
neon = ['Ne', 8]
gases = [helium, neon]
```

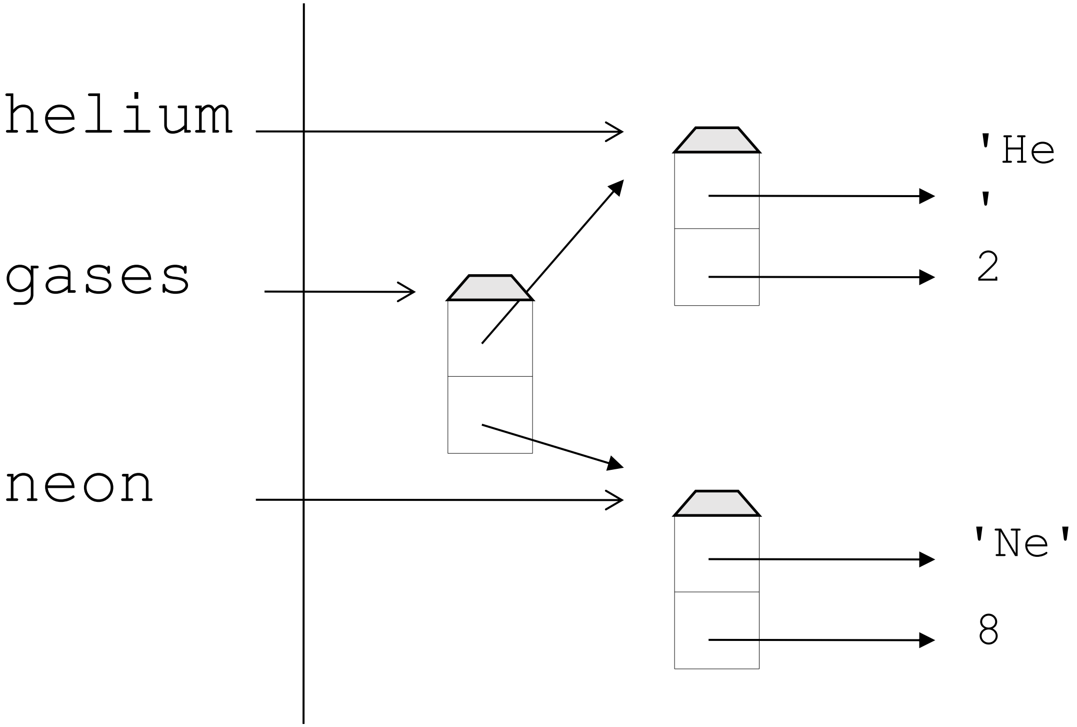
Heterogeneous : can store values of many kinds

```
helium = ['He', 2]
neon = ['Ne', 8]
gases = [helium, neon]
```



Heterogeneous : can store values of many kinds

```
helium = ['He', 2]
neon = ['Ne', 8]
gases = [helium, neon]
```



Devote a whole

episode to this

Loop over elements to "do all"

Loop over elements to "do all"

Use `while` to step through all possible indices

Loop over elements to "do all"

Use `while` to step through all possible indices

```
gases = ['He', 'Ne', 'Ar', 'Kr']
i = 0
while i < len(gases):
    print(gases[i])
    i += 1
```

Loop over elements to "do all"

Use `while` to step through all possible indices

```
gases = ['He', 'Ne', 'Ar', 'Kr']
i = 0
while i < len(gases):
    print(gases[i])
    i += 1
```

First legal index

Loop over elements to "do all"

Use `while` to step through all possible indices

```
gases = ['He', 'Ne', 'Ar', 'Kr']
i = 0
while i < len(gases):
    print(gases[i])
    i += 1
```

Next index

Loop over elements to "do all"

Use `while` to step through all possible indices

```
gases = ['He', 'Ne', 'Ar', 'Kr']
i = 0
while i < len(gases):
    print(gases[i])
    i += 1
```

Defines set of legal indices

Loop over elements to "do all"

Use `while` to step through all possible indices

```
gases = ['He', 'Ne', 'Ar', 'Kr']
i = 0
while i < len(gases):
    print(gases[i])
    i += 1
```

```
He
Ne
Ar
Kr
```

Loop over elements to "do all"

Use `while` to step through all possible indices

```
gases = ['He', 'Ne', 'Ar', 'Kr']
i = 0
while i < len(gases):
    print(gases[i])
    i += 1
He
Ne
Ar
Kr
```

Tedious to type in over and over again

Loop over elements to "do all"

Use `while` to step through all possible indices

```
gases = ['He', 'Ne', 'Ar', 'Kr']
i = 0
while i < len(gases):
    print(gases[i])
    i += 1
He
Ne
Ar
Kr
```

Tedious to type in over and over again

And it's easy to forget the "+= 1" at the end

Use a **`for`** loop to access each value in turn

Use a `for` loop to access each value in turn

```
gases = ['He', 'Ne', 'Ar', 'Kr']
for gas in gases:
    print(gas)
He
Ne
Ar
Kr
```

Use a `for` loop to access each value in turn

```
gases = ['He', 'Ne', 'Ar', 'Kr']
for gas in gases:
    print(gas)
He
Ne
Ar
Kr
```

Loop variable assigned each *value* in turn

Use a `for` loop to access each value in turn

```
gases = ['He', 'Ne', 'Ar', 'Kr']
for gas in gases:
    print(gas)
He
Ne
Ar
Kr
```

Loop variable assigned each *value* in turn

Not each index

Use a `for` loop to access each value in turn

```
gases = ['He', 'Ne', 'Ar', 'Kr']
for gas in gases:
    print(gas)
He
Ne
Ar
Kr
```

Loop variable assigned each *value* in turn

Not each index

Because that's the most common case

Can delete entries entirely (shortens the list)

Can delete entries entirely (shortens the list)

```
gases = ['He', 'Ne', 'Ar', 'Kr']
```

Can delete entries entirely (shortens the list)

```
gases = ['He', 'Ne', 'Ar', 'Kr']
del gases[0]
```

Can delete entries entirely (shortens the list)

```
gases = ['He', 'Ne', 'Ar', 'Kr']
del gases[0]
print(gases)
['Ne', 'Ar', 'Kr']
```

Can delete entries entirely (shortens the list)

```
gases = ['He', 'Ne', 'Ar', 'Kr']
del gases[0]
print(gases)
['Ne', 'Ar', 'Kr']
del gases[2]
```

Can delete entries entirely (shortens the list)

```
gases = ['He', 'Ne', 'Ar', 'Kr']
del gases[0]
print(gases)
['Ne', 'Ar', 'Kr']
del gases[2]
print(gases)
['Ne', 'Ar']
```

Can delete entries entirely (shortens the list)

```
gases = ['He', 'Ne', 'Ar', 'Kr']
del gases[0]
print(gases)
['Ne', 'Ar', 'Kr']
del gases[2]
print(gases)
['Ne', 'Ar']
```

Yes, deleting an index that doesn't exist is an error

Appending values to a list lengthens it

Appending values to a list lengthens it

```
gases = []
```

Appending values to a list lengthens it

```
gases = []
gases.append('He')
```

Appending values to a list lengthens it

```
gases = []
gases.append('He')
gases.append('Ne')
```

Appending values to a list lengthens it

```
gases = []
gases.append('He')
gases.append('Ne')
gases.append('Ar')
```

Appending values to a list lengthens it

```
gases = []
gases.append('He')
gases.append('Ne')
gases.append('Ar')
print(gases)
['He', 'Ne', 'Ar']
```

Appending values to a list lengthens it

```
gases = []
gases.append('He')
gases.append('Ne')
gases.append('Ar')
print(gases)
['He', 'Ne', 'Ar']
```

Most operations on lists are *methods*

Appending values to a list lengthens it

```
gases = []
gases.append('He')
gases.append('Ne')
gases.append('Ar')
print(gases)
['He', 'Ne', 'Ar']
```

Most operations on lists are *methods*

A function that belongs to (and usually operates on)

specific data

Appending values to a list lengthens it

```
gases = []
gases.append('He')
gases.append('Ne')
gases.append('Ar')
print(gases)
['He', 'Ne', 'Ar']
```

Most operations on lists are *methods*

A function that belongs to (and usually operates on)

specific data

```
thing . method (args)
```

Some useful list methods

Some useful list methods

```
gases = ['He', 'He', 'Ar', 'Kr'] # 'He' is duplicated
```

Some useful list methods

```
gases = ['He', 'He', 'Ar', 'Kr'] # 'He' is duplicated
print(gases.count('He'))
2
```

Some useful list methods

```
gases = ['He', 'He', 'Ar', 'Kr'] # 'He' is duplicated
print(gases.count('He'))
2
print(gases.index('Ar'))
2
```

Some useful list methods

```
gases = ['He', 'He', 'Ar', 'Kr'] # 'He' is duplicated
print(gases.count('He'))
2
print(gases.index('Ar'))
2
gases.insert(1, 'Ne')
```

Some useful list methods

```
gases = ['He', 'He', 'Ar', 'Kr'] # 'He' is duplicated
print(gases.count('He'))
2
print(gases.index('Ar'))
2
gases.insert(1, 'Ne')
print(gases)
['He', 'Ne', 'He', 'Ar', 'Kr']
```

Two that are often used incorrectly

Two that are often used incorrectly

```
gases = ['He', 'Ne', 'Ar', 'Kr']
```

Two that are often used incorrectly

```
gases = ['He', 'Ne', 'Ar', 'Kr']
print(gases.sort())
None
```

Two that are often used incorrectly

```
gases = ['He', 'Ne', 'Ar', 'Kr']
print(gases.sort())
None
print(gases)
['Ar', 'He', 'Kr', 'Ne']
```

Two that are often used incorrectly

```
gases = ['He', 'Ne', 'Ar', 'Kr']
print(gases.sort())
None
print(gases)
['Ar', 'He', 'Kr', 'Ne']
print(gases.reverse())
None
```

Two that are often used incorrectly

```
gases = ['He', 'Ne', 'Ar', 'Kr']
print(gases.sort())
None
print(gases)
['Ar', 'He', 'Kr', 'Ne']
print(gases.reverse())
None
print(gases)
['Ne', 'Kr', 'He', 'Ar']
```

Two that are often used incorrectly

```
gases = ['He', 'Ne', 'Ar', 'Kr']
print(gases.sort())
None
print(gases)
['Ar', 'He', 'Kr', 'Ne']
print(gases.reverse())
None
print(gases)
['Ne', 'Kr', 'He', 'Ar']
```

A common bug

Two that are often used incorrectly

```
gases = ['He', 'Ne', 'Ar', 'Kr']
print(gases.sort())
None
print(gases)
['Ar', 'He', 'Kr', 'Ne']
print(gases.reverse())
None
print(gases)
['Ne', 'Kr', 'He', 'Ar']
```

A common bug

`gases = gases.sort()` assigns `None` to `gases`

There is an alternative built-in function for sorting:

```
gases = ['He', 'Ne', 'Ar', 'Kr']
s_gases = sorted(gases)
r_gases = sorted(gases, reverse=True)

print(gases)
['He', 'Ne', 'Ar', 'Kr']

print(s_gases)
['Ar', 'He', 'Kr', 'Ne']

print(r_gases)
['Ne', 'Kr', 'He', 'Ar']
```

Use `in` to test for membership

Use **in** to test for membership

```
gases = ['He', 'Ne', 'Ar', 'Kr']
```

Use **in** to test for membership

```
gases = ['He', 'Ne', 'Ar', 'Kr']
print('He' in gases)
True
```

Use **in** to test for membership

```
gases = ['He', 'Ne', 'Ar', 'Kr']
print('He' in gases)
True
if 'Pu' in gases:
    print('But plutonium is not a gas!')
else:
    print('The universe is well ordered.')
```

Use **in** to test for membership

```
gases = ['He', 'Ne', 'Ar', 'Kr']
print('He' in gases)
True
if 'Pu' in gases:
    print('But plutonium is not a gas!')
else:
    print('The universe is well ordered.')
The universe is well ordered.
```

Use `range` to construct a range of numbers

Use `range` to construct a range of numbers

```
print(range(5))
range(0, 5)
```

Use `list(range)` to construct lists of numbers

Use `list(range)` to construct lists of numbers

```
print(list(range(5)))
[0, 1, 2, 3, 4]
```

Use `list(range)` to construct lists of numbers

```
print(list(range(5)))
[0, 1, 2, 3, 4]
print(list(range(2, 6)))
[2, 3, 4, 5]
```

Use `list(range)` to construct lists of numbers

```
print(list(range(5)))
[0, 1, 2, 3, 4]
print(list(range(2, 6)))
[2, 3, 4, 5]
print(list(range(0, 10, 3)))
[0, 3, 6, 9]
```

Use `list(range)` to construct lists of numbers

```
print(list(range(5)))
[0, 1, 2, 3, 4]
print(list(range(2, 6)))
[2, 3, 4, 5]
print(list(range(0, 10, 3)))
[0, 3, 6, 9]
print(list(range(10, 0)))
[]
```

Sometimes you might need both the index and value while looping

Sometimes you might need both the index and value while looping.

You could use `range(len(gases))`

Sometimes you might need both the index and value while looping.

You could use `range(len(gases))`

```
gases = ['He', 'Ne', 'Ar', 'Kr’]

for i in range(len(gases)):
    print(i, gases[i])
0 He
1 Ne
2 Ar
3 Kr
```

But there is a better way… `enumerate()`

But there is a better way... `enumerate()`

```
gases = ['He', 'Ne', 'Ar', 'Kr’]

for i,gas in enumerate(gases):
    print(i, gas)
0 He
1 Ne
2 Ar
3 Kr
```

But there is a better way... `enumerate()`

```
gases = ['He', 'Ne', 'Ar', 'Kr’]

for i,gas in enumerate(gases):
    print(i, gas)
0 He
1 Ne
2 Ar
3 Kr
```

A very common *idiom* in Python

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…len(X)-1`

Lists, strings, and tuples are all *sequences*

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

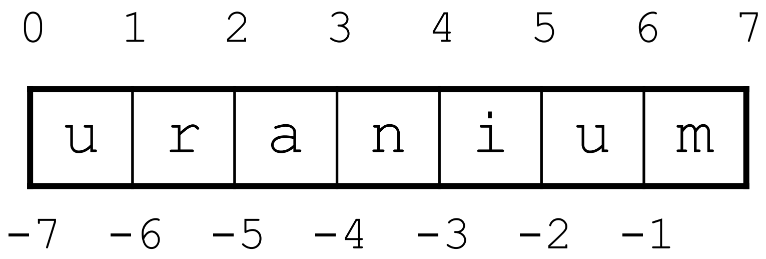
Can also be *sliced* using a range of indices

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'
>>>
```



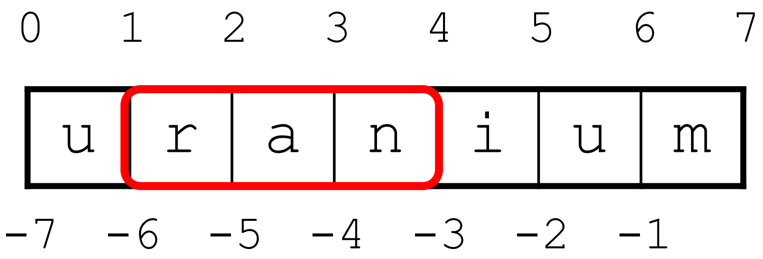
```
0   1   2   3   4   5   6   7
  u   r   a   n   i   u   m
-7  -6  -5  -4  -3  -2  -1
```

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
>>>
```



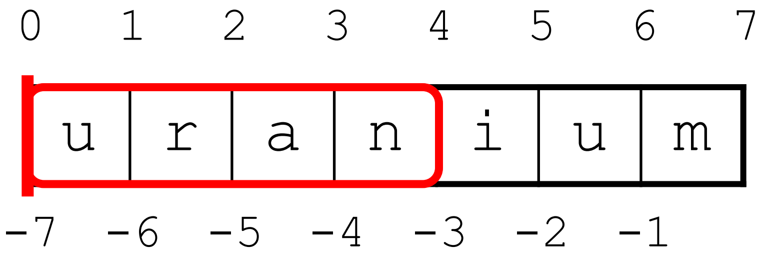
```
 0   1   2   3   4   5   6   7
 | u | r | a | n | i | u | m |
-7  -6  -5  -4  -3  -2  -1
```

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
>>>
```



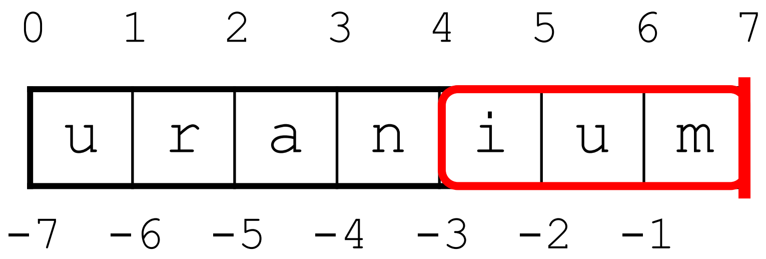
```
0   1   2   3   4   5   6   7
  u   r   a   n   i   u   m
-7  -6  -5  -4  -3  -2  -1
```

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
>>>
```



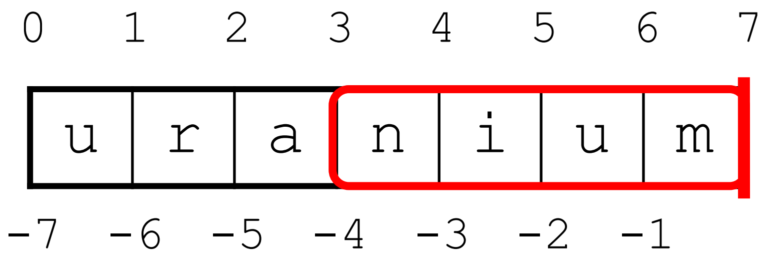
```
 0   1   2   3   4   5   6   7
 | u | r | a | n | i | u | m |
-7  -6  -5  -4  -3  -2  -1
```

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

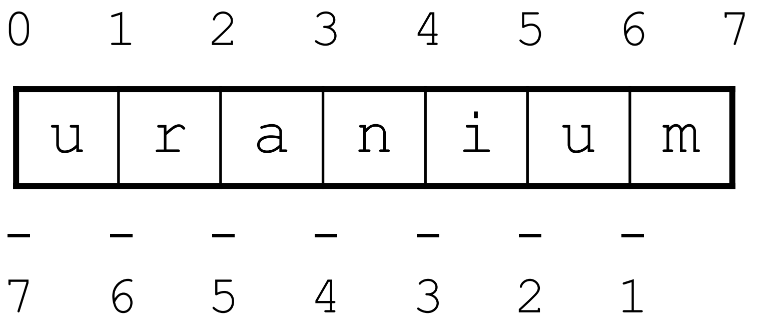
Python checks bounds when indexing

But truncates when slicing

Python checks bounds when indexing

But truncates when slicing

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

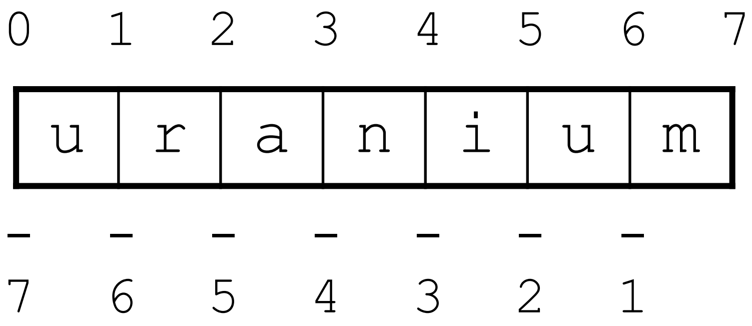


```
0    1    2    3    4    5    6    7
  u    r    a    n    i    u    m
-    -    -    -    -    -    -
7    6    5    4    3    2    1
```

Python checks bounds when indexing

But truncates when slicing

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



0 1 2 3 4 5 6 7

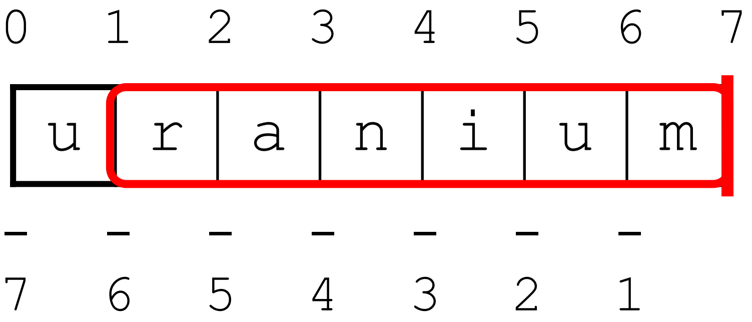
u r a n i u m

-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
>>>
```



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

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

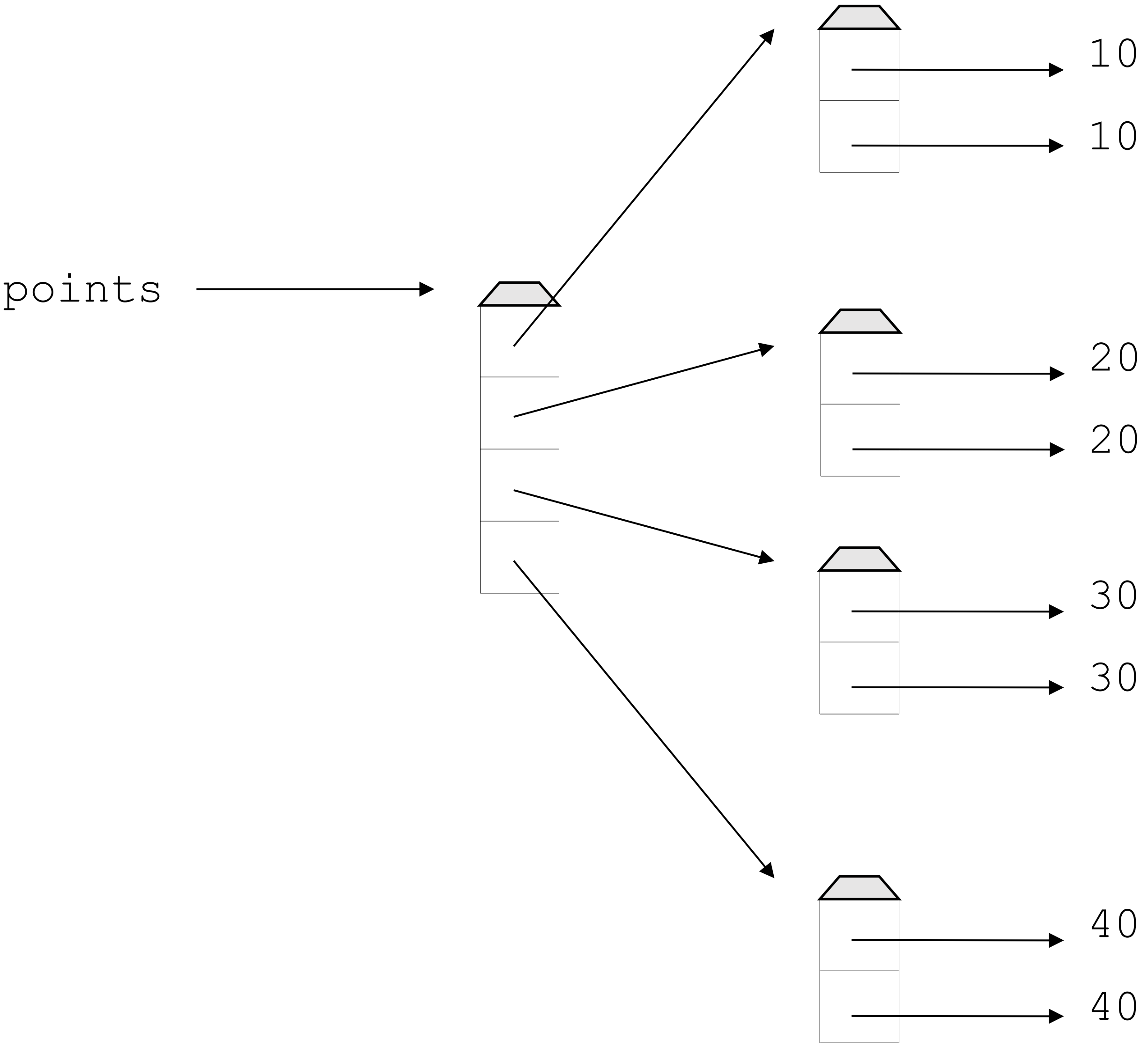
```
>>> points = [[10, 10], [20, 20], [30, 30], [40, 40]]
>>> middle = points[1:-1]
>>> middle[0][0] = 'whoops'
>>> middle[1][0] = 'aliasing'
>>> print(middle)
[['whoops', 20], ['aliasing', 30]]
>>>
```

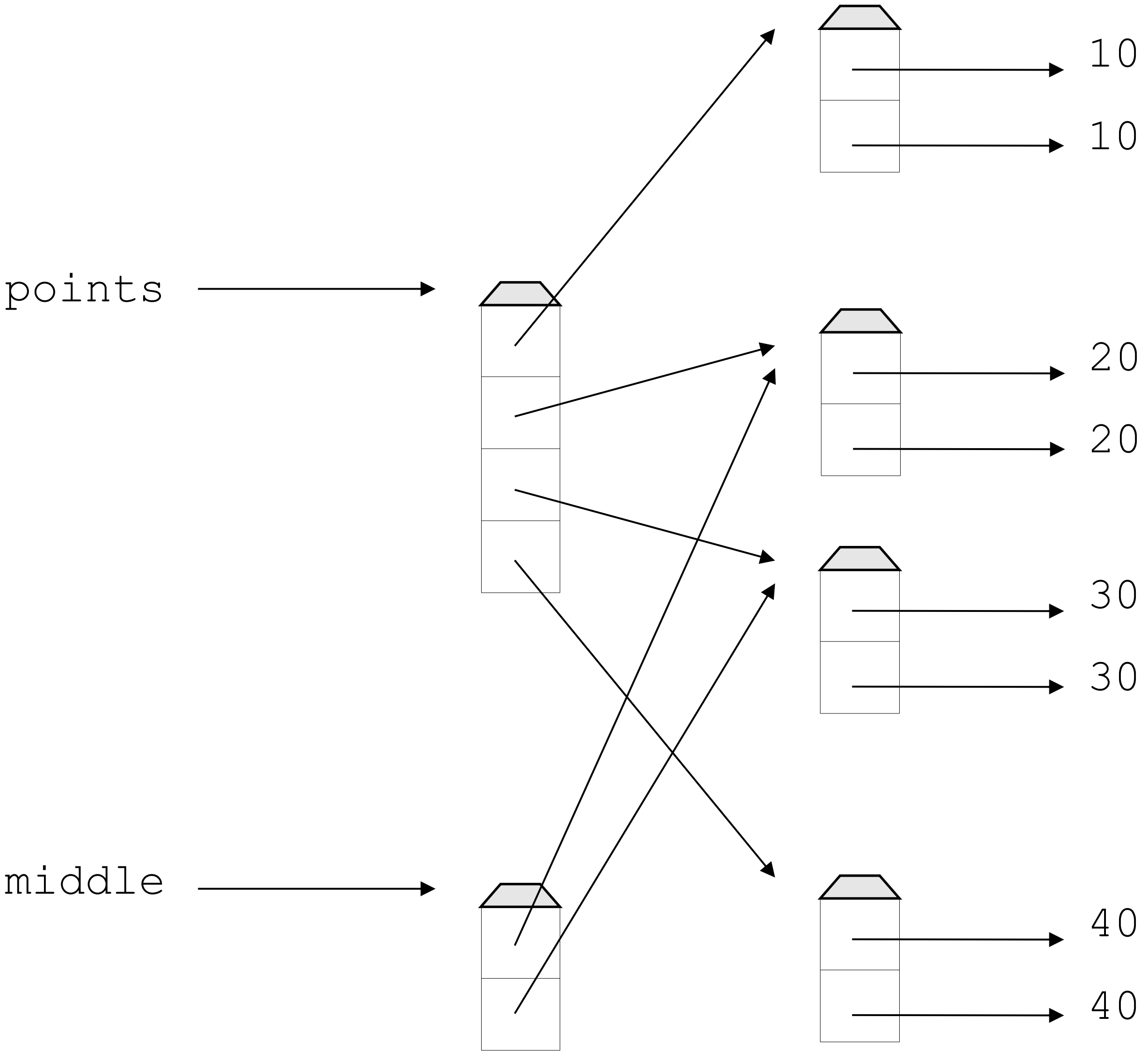
Slicing always creates a new collection

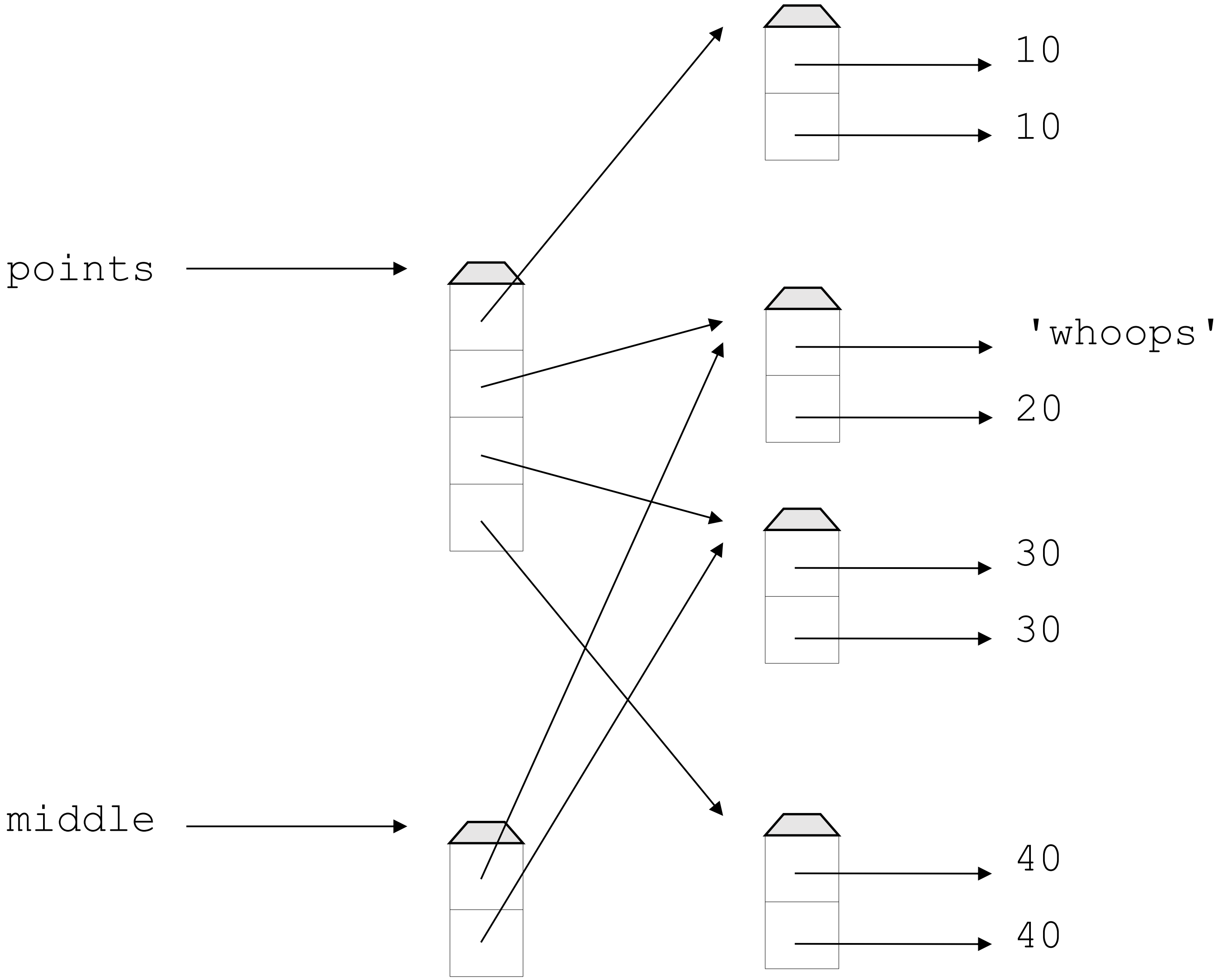
Beware of aliasing

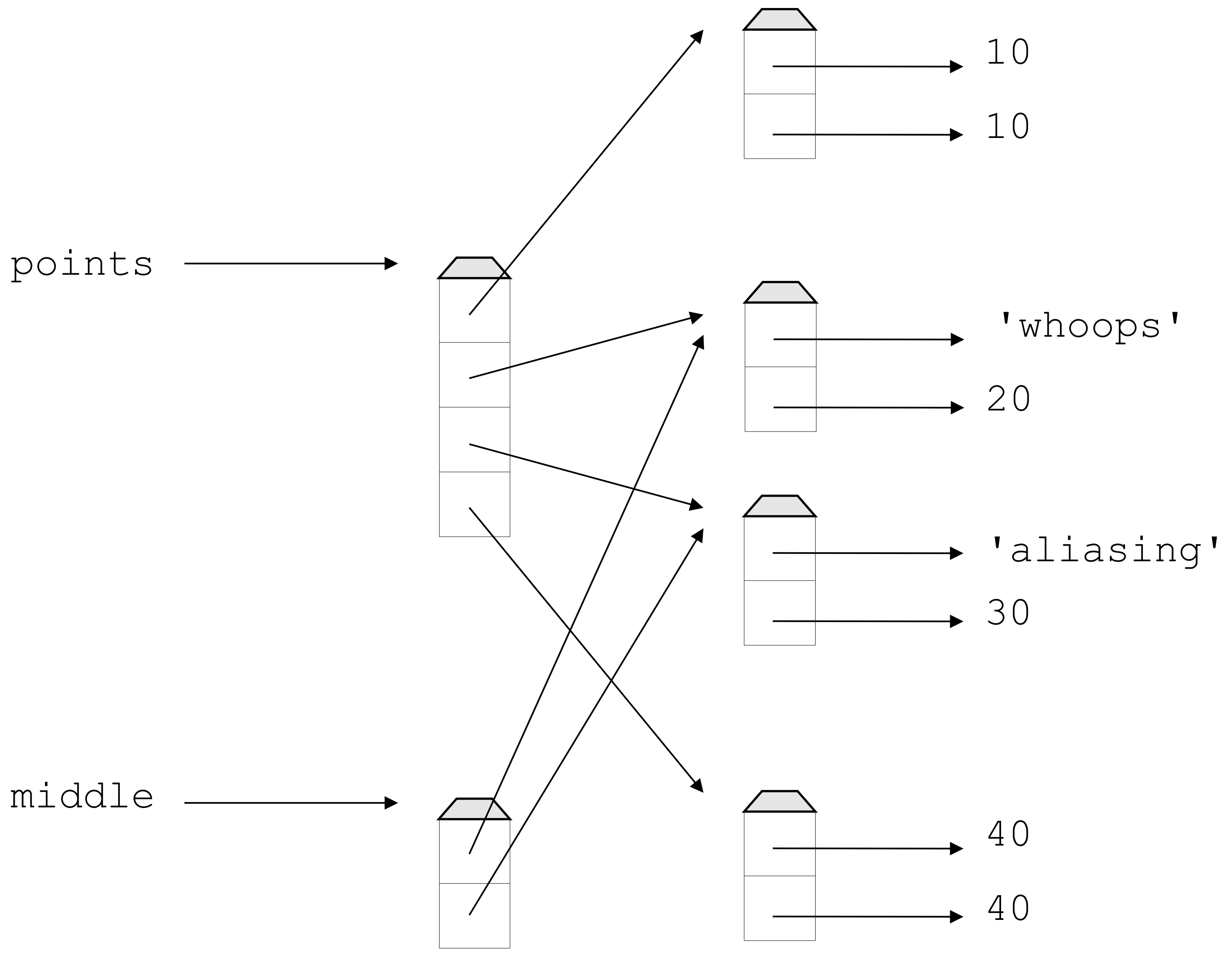
```
>>> points = [[10, 10], [20, 20], [30, 30], [40, 40]]
>>> middle = points[1:-1]
>>> middle[0][0] = 'whoops'
>>> middle[1][0] = 'aliasing'
>>> print(middle)
[['whoops', 20], ['aliasing', 30]]
>>> print(points)
[[10, 10], ['whoops', 20], ['aliasing', 30], [40, 40]]
>>>
```

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
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