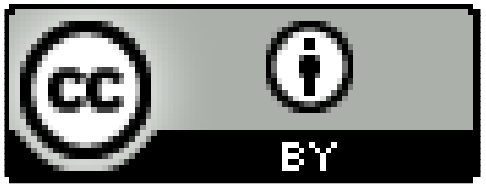
Python

Tuples



A *list* is a mutable heterogeneous sequence

A *list* is a mutable heterogeneous sequence

A *tuple* is an *immutable* heterogeneous sequence

A *list* is a mutable heterogeneous sequence

A *tuple* is an *immutable* heterogeneous sequence

I.e., a list that can't be changed after creation

A *list* is a mutable heterogeneous sequence

A *tuple* is an *immutable* heterogeneous sequence

I.e., a list that can't be changed after creation

Why provide a less general type of collection?

A *list* is a mutable heterogeneous sequence

A *tuple* is an *immutable* heterogeneous sequence

I.e., a list that can't be changed after creation

Why provide a less general type of collection?

Full explanation will have to wait for lecture on sets and dictionaries

A *list* is a mutable heterogeneous sequence

A *tuple* is an *immutable* heterogeneous sequence

I.e., a list that can't be changed after creation

Why provide a less general type of collection?

Full explanation will have to wait for lecture on sets and dictionaries

Useful even before then

Create tuples using `()` instead of `[]`

Create tuples using `()` instead of `[]`

Still index using `[]` (because everything does)

Create tuples using `()` instead of `[]`

Still index using `[]` (because everything does)

```
>>> primes = (2, 3, 5, 7)
>>> print primes[0], primes[-1]
2 7
>>>
```

Create tuples using `()` instead of `[]`

Still index using `[]` (because everything does)

```
>>> primes = (2, 3, 5, 7)
>>> print primes[0], primes[-1]
2 7
>>> empty_tuple = ()
>>> print len(empty_tuple)
0
>>>
```

Create tuples using `()` instead of `[]`

Still index using `[]` (because everything does)

```
>>> primes = (2, 3, 5, 7)
>>> print primes[0], primes[-1]
2 7
>>> empty_tuple = ()
>>> print len(empty_tuple)
0
>>>
```

Must use `(val,)` for tuple with one element

Create tuples using `()` instead of `[]`

Still index using `[]` (because everything does)

```
>>> primes = (2, 3, 5, 7)
>>> print primes[0], primes[-1]
2 7
>>> empty_tuple = ()
>>> print len(empty_tuple)
0
>>>
```

Must use `(val,)` for tuple with one element

Because math says that `(5)` is just 5

Create tuples using `()` instead of `[]`

Still index using `[]` (because everything does)

```
>>> primes = (2, 3, 5, 7)
>>> print primes[0], primes[-1]
2 7
>>> empty_tuple = ()
>>> print len(empty_tuple)
0
>>>
```

Must use `(val,)` for tuple with one element

Because math says that `(5)` is just 5

One of Python's few syntactic warts...

Don't need parentheses if context is enough

Don't need parentheses if context is enough

```
>>> primes = 2, 3, 5, 7
>>> print primes
(2, 3, 5, 7)
>>>
```

Don't need parentheses if context is enough

```
>>> primes = 2, 3, 5, 7
>>> print primes
(2, 3, 5, 7)
>>>
```

Can use on the left of assignment

Don't need parentheses if context is enough

```
>>> primes = 2, 3, 5, 7
>>> print primes
(2, 3, 5, 7)
>>>
```

Can use on the left of assignment

```
>>> left, middle, right = 2, 3, 5
>>>
```

Don't need parentheses if context is enough

```
>>> primes = 2, 3, 5, 7
>>> print primes
(2, 3, 5, 7)
>>>
```

Can use on the left of assignment

```
>>> left, middle, right = 2, 3, 5
>>> print left
2
>>> print middle
3
>>> print right
5
>>>
```

Don't need parentheses if context is enough

```
>>> primes = 2, 3, 5, 7
>>> print primes
(2, 3, 5, 7)
>>>
```

Can use on the left of assignment

```
>>> left, middle, right = 2, 3, 5
>>> print left
2
>>> print middle
3
>>> print right
5
>>>
```

With great power comes great responsibility...

Allows functions to return multiple values

Allows functions to return multiple values

```
>>> def bounds(values):
...     low = min(values)
...     high = max(values)
...     return (low, high)
...
>>>
```

Allows functions to return multiple values

```
>>> def bounds(values):
...     low = min(values)
...     high = max(values)
...     return (low, high)
...
>>> print bounds([3, -5, 9, 4, 17, 0])
(-5, 17)
>>>
```

Allows functions to return multiple values

```
>>> def bounds(values):
...     low = min(values)
...     high = max(values)
...     return (low, high)
...
>>> print bounds([3, -5, 9, 4, 17, 0])
(-5, 17)
>>> least, greatest = bounds([3, -5, 9, 4, 17, 0])
>>> print least
5
>>> print greatest
17
>>>
```

Sometimes used to return (success, result) pairs

Sometimes used to return (success, result) pairs

```
def read_if_available(datafile_name):
  if file_exists(datafile_name):
    ...
    return (True, data_values)
  else:
    return (False, [])
```

Sometimes used to return (success, result) pairs

```
def read_if_available(datafile_name):
  if file_exists(datafile_name):
    ...
    return (True, data_values)
  else:
    return (False, [])

success, data = read_if_available('mydata.dat')
if success:
  ...
```

Sometimes used to return (success, result) pairs

```
def read_if_available(datafile_name):
  if file_exists(datafile_name):
    ...
    return (True, data_values)
  else:
    return (False, [])

success, data = read_if_available('mydata.dat')
if success:
  ...
```

We'll meet a better way in the lecture on testing

Provides a quick way to swap variables' values

Provides a quick way to swap variables' values

```
>>> left, right = 0, 10
>>>
```

Provides a quick way to swap variables' values

```
>>> left, right = 0, 10
>>> right, left = left, right
>>>
```

Provides a quick way to swap variables' values

```
>>> left, right = 0, 10
>>> right, left = left, right
>>> print right
0
>>> print left
10
>>>
```

Provides a quick way to swap variables' values

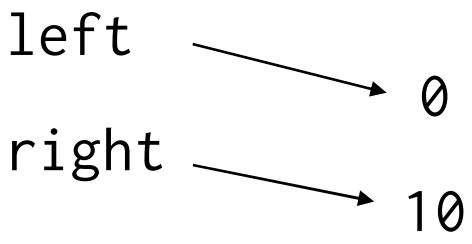
```
>>> left, right = 0, 10
>>> right, left = left, right
>>> print right
0
>>> print left
10
>>>
```

Python creates temporaries if needed

Provides a quick way to swap variables' values

```
>>> left, right = 0, 10
>>> right, left = left, right
>>> print right
0
>>> print left
10
>>>
```

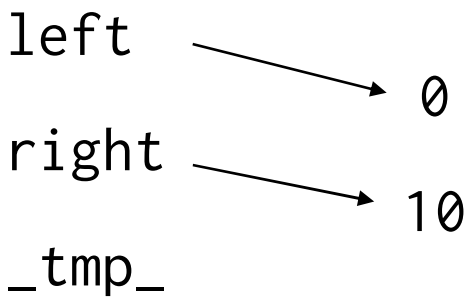
Python creates temporaries if needed



Provides a quick way to swap variables' values

```
>>> left, right = 0, 10
>>> right, left = left, right
>>> print right
0
>>> print left
10
>>>
```

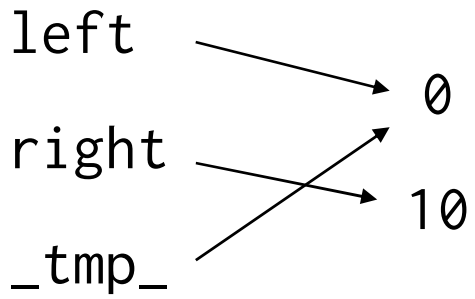
Python creates temporaries if needed



Provides a quick way to swap variables' values

```
>>> left, right = 0, 10
>>> right, left = left, right
>>> print right
0
>>> print left
10
>>>
```

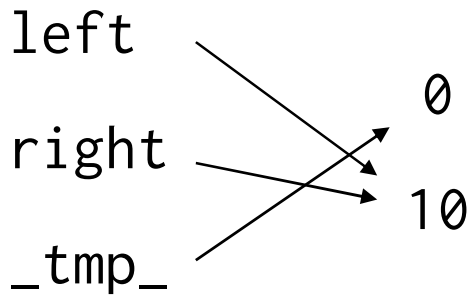
Python creates temporaries if needed



Provides a quick way to swap variables' values

```
>>> left, right = 0, 10
>>> right, left = left, right
>>> print right
0
>>> print left
10
>>>
```

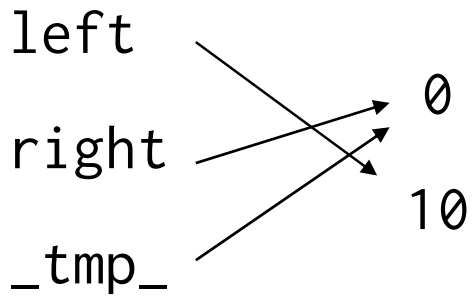
Python creates temporaries if needed



Provides a quick way to swap variables' values

```
>>> left, right = 0, 10
>>> right, left = left, right
>>> print right
0
>>> print left
10
>>>
```

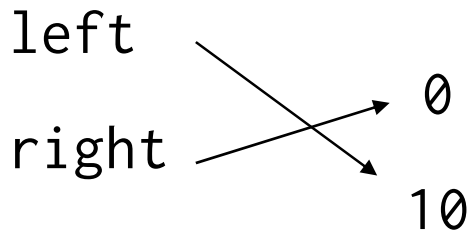
Python creates temporaries if needed



Provides a quick way to swap variables' values

```
>>> left, right = 0, 10
>>> right, left = left, right
>>> print right
0
>>> print left
10
>>>
```

Python creates temporaries if needed



And an easy way to unpack a list

And an easy way to unpack a list

```
>>> colors = ['yellow', 'magenta', 'lavender']
>>>
```

And an easy way to unpack a list

```
>>> colors = ['yellow', 'magenta', 'lavender']
>>> left, middle, right = colors
>>>
```

And an easy way to unpack a list

```
>>> colors = ['yellow', 'magenta', 'lavender']
>>> left, middle, right = colors
>>> print left
yellow
>>> print middle
magenta
>>> print right
lavender
>>>
```

And an easy way to unpack a list

```
>>> colors = ['yellow', 'magenta', 'lavender']
>>> left, middle, right = colors
>>> print left
yellow
>>> print middle
magenta
>>> print right
lavender
>>>
```

Number of values must be the same

Often used in loops

Often used in loops

```
>>> pairs = ((1, 10), (2, 20), (3, 30), (4, 40))
>>>
```

Often used in loops

```
>>> pairs = ((1, 10), (2, 20), (3, 30), (4, 40))
>>> for p in pairs:
...   print p[0] + p[1]
```

Often used in loops

```
>>> pairs = ((1, 10), (2, 20), (3, 30), (4, 40))
>>> for p in pairs:
...   print p[0] + p[1]
```

Often used in loops

```
>>> pairs = ((1, 10), (2, 20), (3, 30), (4, 40))
>>> for (low, high) in pairs:
...   print low + high
```

Often used in loops

```
>>> pairs = ((1, 10), (2, 20), (3, 30), (4, 40))
>>> for (low, high) in pairs:
...     print low + high
...
11
22
33
44
>>>
```

The `enumerate` function produces (index, value) pairs

The `enumerate` function produces (index, value) pairs

```
>>> colors = ['yellow', 'magenta', 'lavender']
>>> for (i, name) in enumerate(colors):
...   print i, name
```

The `enumerate` function produces (index, value) pairs

```
>>> colors = ['yellow', 'magenta', 'lavender']
>>> for (i, name) in enumerate(colors):
...     print i, name
...
0 yellow
1 magenta
2 lavender
>>>
```

The `enumerate` function produces (index, value) pairs

```
>>> colors = ['yellow', 'magenta', 'lavender']
>>> for (i, name) in enumerate(colors):
...   print i, name
...
0 yellow
1 magenta
2 lavender
>>>
```

Prefer this to `range(len(values))`

created by

Greg Wilson

October 2010