TUPLES

As Data Structures

Tuples vs Lists vs Strings

Tuples	Lists	Strings
containers	containers	containers
order matters	order matters	order matters
Heterogeneous / Homogeneous	Heterogeneous / Homogeneous	Homogeneous
indexable	indexable	indexable
iterable	iterable	iterable
immutable	mutable	immutable
fixed length	length can change	fixed length
fixed order	order of elements can change	fixed order
cannot do in-place sorts	can do in-place sorts	
cannot do in-place reversals	can do in-place reversals	

Immutability of Tuples

elements cannot be added or removed

the order of elements cannot be changed

works well for representing data structures:

Point: `(10, 20)`

1st element is the x-coordinate

2nd element is the y-coordinate

Circle: `(0, 0, 10)`

1st element is the x-coordinate of the center
2nd element is the y-coordinate of the center
3rd element is the radius

City: `('London', 'UK', 8_780_000)`

1st element is the name of a city
2nd element is the country
3rd element is the population

The position of the data has meaning

Tuples as Data Records

Think of a tuple as a data record where the position of the data has meaning

```
london = ('London', 'UK', 8_780_000)

new_york = ('New York', 'USA', 8_500_000)

beijing = ('Beijing', 'China', 21_000_000)
```

Because tuples, strings and integers are immutable, we are guaranteed that the data and data structure for `london` will never change

We can have a list of these tuples:

```
cities = [('London', 'UK', 8_780_000),
          ('New York', 'USA', 8_500_000),
          ('Beijing', 'China', 21_000_000)]
```

Extracting data from Tuples

Since tuples are sequences just like strings and lists, we can retrieve items by index

```
london = ('London', 'UK', 8_780_000)

city = london[0]        country = london[1]        population = london[2]
```

```
cities = [('London', 'UK', 8_780_000),
          ('New York', 'USA', 8_500_000),
          ('Beijing', 'China', 21_000_000)]
```

```
total_population = 0
for city in cities:
    total_population += city[2]
```

You'll notice how the list of cities is homogeneous (contains cities only)

But a city (the tuple) is heterogeneous

Extracting data from Tuples

We can also use tuple unpacking

We actually already know how to do this – we covered this in the section on function arguments

```
new_york = ('New York', 'USA', 8_500_000)
```

packed three values into a tuple

```
city, country, population = new_york
```

unpacked tuple

```
city, country, population = ('New York', 'USA', 8_500_000)
```

```
city, country, population = 'New York', 'USA', 8_500_000
```

Dummy Variables

This is something you're likely to run across when you look at Python code that uses tuple unpacking

Sometimes, we are only interested in a subset of the data fields in a tuple, not all of them

Suppose we are interested only in the city name and the population:

```
city, _, population = ('Beijing', 'China', 21_000_000)
```

_ is actually a legal variable name – so there's nothing special about it

but by convention, we use the underscore to indicate this is a variable we don't care about

in fact, we could just have used:

```
city, ignored, population = ('Beijing', 'China', 21_000_000)
```

Dummy Variables

It's also used in extended unpacking too

```
record = ('DJIA', 2018, 1, 19, 25987.35, 26071.72, 25942.83, 26071.72)
symbol, year, month, day, open, high, low, close = record
```

Let's say we are only interested in the `symbol`, `year`, `month`, `day` and `close` fields

We could do it this way:

```
symbol = record[0]
year = record[1]
month = record[2]
day = record[3]
close = record[7]
```

looks really bad!

```
symbol, year, close = record[0], record[1], record[7]
```

awful!

```
symbol, year, month, day, *_, close = record

symbol, year, month, day, *ignored, close = record
```

Code