Tuples

See Section 6.4 of the Notes

We know lists are *mutable* structures. This means the data in a list can be changed after the list is created. For example, the following code:

```
L = [1, 2, 3]
L[1] = "two"
```

changes list L to [1, "two", 3]. Strings aren't mutable; you will get an error message from the following code:

```
s = "bob"
s[1] = 'u'
```

Tuples are **immutable** structures similar to lists. Instead of square brackets, tuples use round backets -- parentheses. For example, (1, 2, 3) is a tuple with 3 elements. (2,) is a tuple with just one element. ( ) is the empty tuple with no elements.

What will this print?

```
def main():
    T = (1, 2, 3)
    foo(T)
    print(T)

def foo(T):
    for i in T:
        print(i)
```

A

1
2
3
(1, 2, 3)

B

1
2
3

C

(1, 2, 3)

D

Nothing; it causes an error.

What will this print?

```
def main():
    T = (1, 2, 3)
    foo2(T)
    print(T)

def foo2(T):
    T = (4, 5)
    return T
```

A	B	C	D
(1, 2, 3)	(4, 5)	(4, 5) (1, 2, 3)	Nothing; it causes an error.

What will this print?

```
def main():
    T = (1, 2, 3)
    foo3(T)
    print(T)

def foo3(T):
    T[0] = 34
```

A	B	C	D
(34, 2, 3)	(1, 2, 3)	(34, 2, 3)	It causes an error

Why, and when, would you use tuples instead of lists? Here are two situations:

a) Sometimes you need immutable types. For example, the keys of a dictionary must be immutable. You can't use lists as dictionary keys, but you can use tuples.
b) Tuples are simpler and take up less memory space than lists. If you have a program that stores lots of points with (x,y) coordinates, it is more efficient to store them as tuples than as lists.