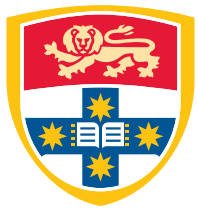
Introduction to Programming (Adv)

School of Computer Science, University of Sydney



Copyright Warning

Tuples

An immutable collection

Tuples: a restricted list

There are cases where there is no need to modify the contents of a list, or it's length.

List is mutable, contents can change []
Tuple is immutable, contents cannot change ()

Returning from a function is a good example of this.

- When the function returns, it must hand over control of the object to the caller. The function then ceases to exist.
- The information returned are values as an output of the function.
- The order in which those return values are presented is fixed.

These restrictions are useful for enforcing *safety* when dealing with packing and unpacking of sequence data.

Tuples: a restricted list (cont.)

Recall `enumerate()`. It will provide an index and the value of each element.

```
1  words = ['book', 'lamp', 'desk', 'chair', 'pen']
2  enum_words = list(enumerate(words))
3  i = 0
4  while i < len(enum_words):
5      item = enum_words[i]
6      print(i, " item: ", item)
7      print(i, " item[0]: ", item[0])
8      print(i, " item[1]: ", item[1])
9      i = i + 1
```

Tuples: a restricted list (cont.)

For this to work, there needs to be a match between the input mapping and the output mapping.

> Sequence unpacking requires that there are as many variables on the left side of the equals sign as there are elements in the sequence.

Programmer *expects* enumerate to return in this form:

```
[(0, 'book'), (1,'lamp'), (2,'desk'), (3,'chair'), (4,'pen')]
```

Tuples: a restricted list (cont.)

Programmer can pass an object without worrying about changes. They are sharing the object, sharing that memory region

```
1  def untrusted_code_plugin(stats):
2      (name, score) = stats
3
4      # hmm, according to documentation, 2nd element is the score...
5      # I try to modify server data so I am the best player there is
           .
6      stats[1] = 999999
7      stats[0] = stats[0] + " is the best you know"
```

```
1  def untrusted_code_plugin(account_details):
2      (account, balance) = account_details
3
4      # hmm, according to docs, this function is called every day
5      # I will add a bit of interest to my offshore bank balance
           hehe
6      if account == 9293728349239O228:
7          account_details[1] *= 1.01
```

Tuples: a restricted list (cont.)

```
1 s = (1, 2, 3)
2 s[1] = 4
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Is this a tuple?

Suppose we have the following code. Identify where there is a tuple, and how many objects.

```
1 x = 1, 2, 3
```

```
1 x = (1, 2, 3)
```

Is this a tuple? (cont.)

```
1 a, b, c, = 1, 2, 3
```

```
1 a, b, c, = (1, 2, 3)
```

How can you deduce it?

Is this a tuple? (cont.)

Functions follow the rule: always return one object.

```
1 def foo():
2     return 1, 2
3
4 # reason ?
5 ret = foo()
```

```
1 def foo():
2     return (1, 2)
3
4 # reason ?
5 ret = foo()
```

Is this a tuple? (cont.)

Trickier

```
1  def foo():
2      return x, 1, (2, 3), [ y, (3, 2), x ]
3
4  # reason ?
5  ret = foo()
```

```
1  def foo():
2      return (1, 2)
3
4  # reason ?
5  a, b = foo()
```

Immutable vs mutable objects

Immutable

- int, float, bool, string are never updated, there is a new object each time.

```
1 x = 5000
2 print(id(x))
3 x += 1
4 print(id(x))
```

- Tuple can store multiple elements, has an index, cannot change

Immutable vs mutable objects (cont.)

Every change to an immutable type can force the construction of a new immutable type.

How many objects?

```
1  def add(x,y):
2      total = x + y
3      return total
4
5  a = 500
6  b = 800
7  total = add(a, b)
```

How many objects?

```
5  a = 500
6  b = 800
7  total = 0
8  while a < 800:
9      total += add(a, b)
10     a += 1
```

Immutable vs mutable objects (cont.)

Mutable

- List can store multiple elements, has an index, can change
- Set can store multiple elements, no order, no duplicates, can change
- Dictionaries can store multiple elements, duplicates have a key, can change
- File objects can be modified, though we will see this next week!

Immutable vs mutable objects (cont.)

```
1  def add(x, y, total_list):
2      total_list[0] += x + y
3
4  a = 500
5  b = 800
6  total_list = [0]
7  while a < 800:
8      add(a, b, total_list)
9      a += 1
10
11 print(total_list[0])
```

Remember, scope matters whether immutable or mutable.