Lecture 7a
Data type - Tuples

What is a Tuple?

A tuple is a sequence of immutable Python objects.

`tup1 = ()` is an empty tuple

`tup2 = (1,)` contains only one element in the tuple. Not `(1)` !!!

```
>>> type((1,)
>>> type((1))
```

Why use a tuple instead of a list?

- Program execution is faster when manipulating a tuple than a list.
- When the data is not to be modified.

Indexing and Slicing

```
tup = (‘a’, ‘b’, ‘c’, ‘d’,
‘e’, ‘f’, ‘g’)

tup[2] = ‘c’
tup[2:5] = (‘c’, ‘d’, ‘e’)
```

Tuples are immutable.

```
tup = ('a', 'b', 'c', 'd',
'e', 'f', 'g')
```

Cannot update a value in the tuple by assignment

```
tup[2] = 'h'
```

Tuples are immutable.

```
tup = ('a', 'b', 'c', 'd',
'e', 'f', 'g')
```

Cannot add any element into a tuple

```
tup.append('h')
```

Tuples are immutable.

```
tup = ('a', 'b', 'c', 'd',
'e', 'f', 'g')
```

Cannot remove any element in a tuple

```
del tup[2]
tup.remove('g')
```

Tuples are immutable.

```
tup = (‘a’, ‘b’, ‘c’, ‘d’,
‘e’, ‘f’, ‘g’)
```

Can delete the whole tuple

```
del tup
```

Concatenation +

```
tup1 = ('a', 'b')
>>> id(tup1)

>>> tup1 = tup1 + (1, 2)
>>> tup1
>>> id(tup1)
```

A new tuple is created !

Repetition *

```
tup = ('a', 'b', 'c')

>>> tup * 3
('a', 'b', 'c', 'a', 'b',
'c', 'a', 'b', 'c')
```

Membership

```
tup = ('a', 'b', 'c')

>>> 'b' in tup
True
```

Iteration

```
tup = ('a', 'b', 'c')

for ele in tup:
    print(ele)
```

```
'a'
'b'
'c'
```

Built-in Tuple Functions

```
string = ‘aAbz’

>>> tup = tuple(string)
tup = ('a', 'A', 'b', 'Z')

>>> len(tup)
>>> max(tup)
>>> min(tup)
```

Built-in Tuple Functions

```
tup = (5, 2, 3)

>>> id(tup)      2121760223304

>>> sorted(tup)
>>> tup           [2, 3, 5]
>>> id(tup)      2121760223304
```

It became a list!

Tuple as a Return value

```
def test();
    a, b = 1, 2
    return a, b

>>> test()
```

The output will be a tuple.

Use of Tuples

Function can only return a single value, but by making that value a tuple, we can effectively group together as many values as we like, and return them together.

```
tup = (1,2,3,4,5)

def score(tup):
    high = max(tup)
    low = min(tup)
return (high, low)
```

With or without the bracket, the output will still be a tuple.

Use of Tuples

Other examples of tuple outputs :

```
return (mean, std_dev)
return (year, mth, day)
return (rabbit_num, wolf_num)
```

Use of Tuples

For example, we could write a function that returns both the area and the circumference of a circle of radius r:

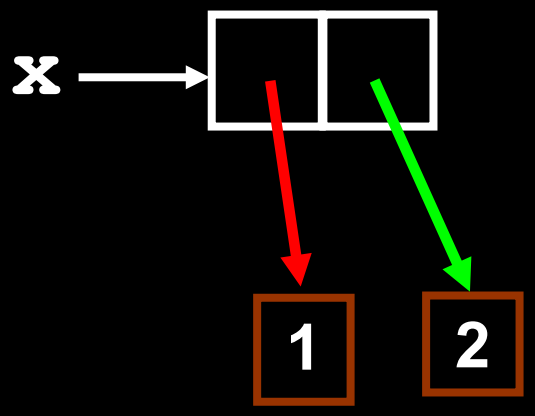
```
import math
def f(r):
    c = 2 * math.pi * r
    a = math.pi * r * r
    return c, a
```

Box-and-Pointer

```
x = (1, 2)

>>> x → (1, 2)

>>> x[0] → 1
>>> x[1] → 2
```



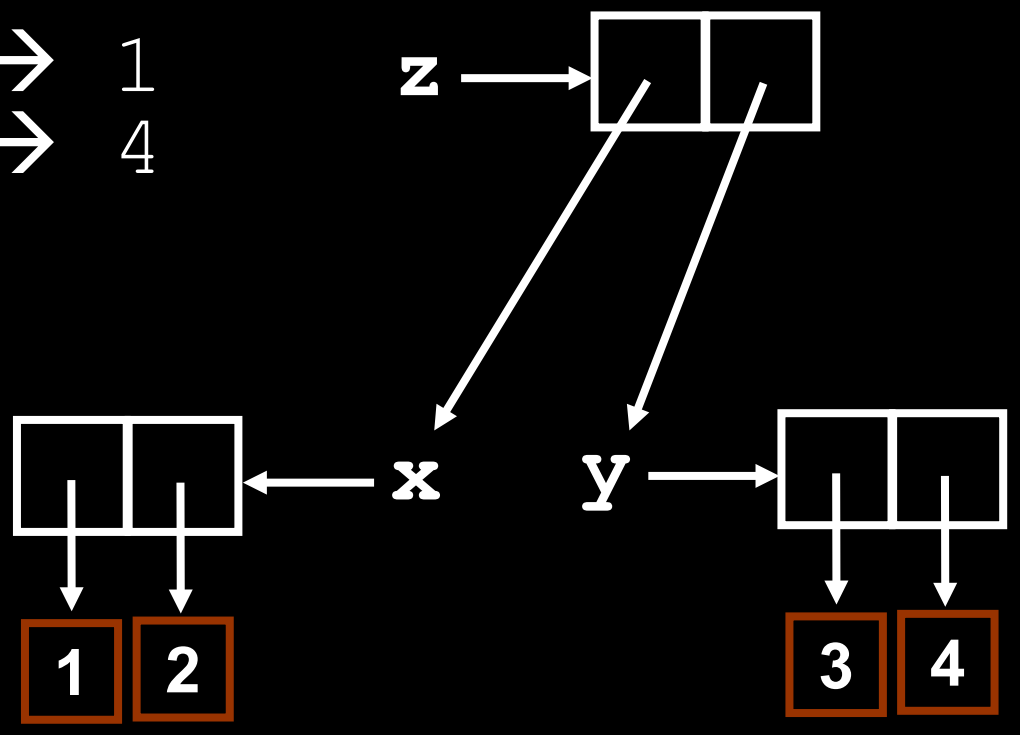
- Variable `x` points to tuple
- Left arrow is x[0]
- Right arrow is x[1]
- Numbers are ***outside*** the tuple, not inside

Box-and-Pointer

```
x = (1, 2)
y = (3, 4)

z = (x, y) # A tuple of tuples

>>> z[0][0] → 1
>>> z[1][1] → 4
```

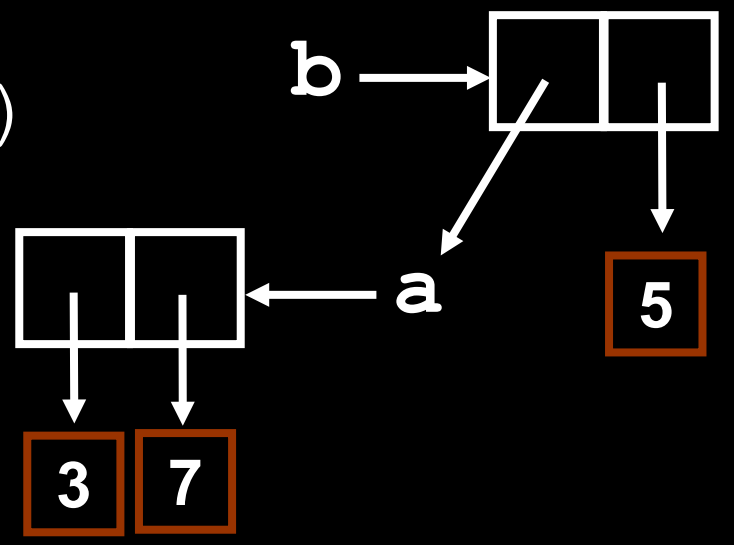


Tuple of Tuples

```
a = (3, 7)
>>> a → (3, 7)

b = (a, 5)
>>> b → ((3, 7), 5)

>>> b[0][0] → 3
>>> b[0][1] → 7
```



Equality

What does equality mean?

Two possibilities (usually)

1. Identity / Identical (‘is’)
   - This means the SAME object (reference in memory)
   - In Python, we use **`is`** to test for this.

Two possibilities (usually)

2. Equivalence ('==')
   - This means two objects that are equivalent (even if they are not the same object)
   - In Python, we use **==** to test for this.

Identity is not the same as Equivalence

Equality

`is` returns **`True`** if the two objects are the same object

`==` returns **`True`** if the two objects are equivalent

```
a = 1
b = 2
x = (a, b)
y = (a, b)
>>> x is y → False
>>> x == y → True
z = x
>>> z is x → True
>>> z is y → False
>>> z == y → True
```

Caution with **`is`**

`is` cannot be used to compare numbers reliably.

```
>>> 3 is 3
⇒ True

>>> 3.000 is 3
⇒ False
```

Moral of the story

Use `==` and `is` carefully, so to save yourself grief.