Foundations of Programming

The Turtle, Lists and Tuples

Announcements

- Lab03 is released. We'll preview it today.

Learning outcomes/key ideas

- Write more complex functions in Python

- Explain how the Turtle works in Python

- Explain the concept of a reference and draw memory model diagrams that use references

- Use the data types list and tuple

Python's Turtle Package

```
import turtle

turtle.forward(100)
turtle.right(90)
turtle.backward(200)
```

Python's Turtle Package

```
import turtle

turtle.forward(100)
turtle.right(90)
turtle.backward(200)
print(turtle.position())
```

What happens if I remove the parentheses from turtle.position() on the last line? I.e.

```
print(turtle.position)
```

Turtles, Turtles everywhere (and introducing references)

```
import turtle

firstTurtle = turtle.Turtle()
secondTurtle = turtle.Turtle()
firstTurtle.forward(100)
secondTurtle.right(90)
secondTurtle.backward(200)
```
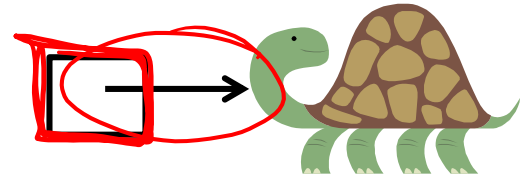
Creates a new turtle object

Creates a new turtle object

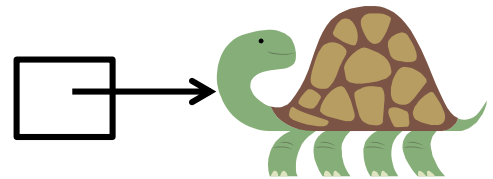**Object oriented Programming**

object: packaged up state, with associated functions

firstTurtle

secondTurtle

Stores a reference to a turtle object

Two different turtle "objects" with the same capabilities
The variables store "references" (location of the Turtle in memory) to each Turtle.

Turtles, Turtles everywhere (and introducing references)

```
import turtle

firstTurtle = turtle.Turtle()
secondTurtle = turtle.Turtle()
secondTurtle = firstTurtle
firstTurtle.forward(100)
secondTurtle.right(90)
secondTurtle.backward(200)
```

firstTurtle

secondTurtle
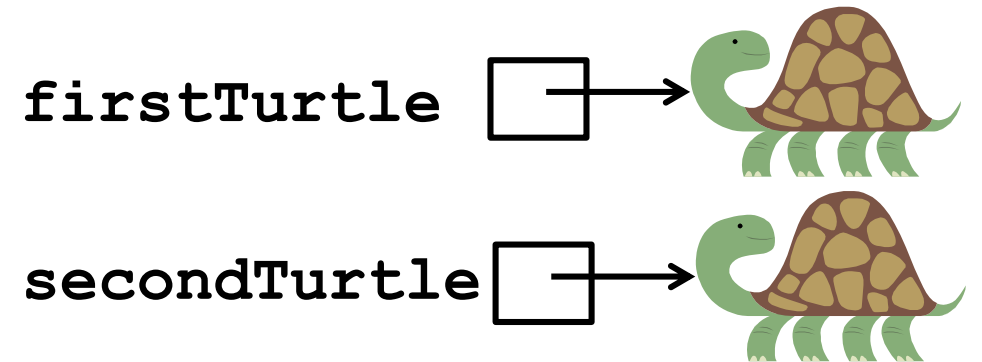
How does the diagram change with the line in red? How does that change what is drawn?

Memory Models in Python, revisited
everything in Python is a reference!

```
x = 42

y = 75

y = x

x = 101
```

```
maria = turtle.Turtle()
jose = turtle.Turtle();
maria = jose;
jose.forward(100);
```
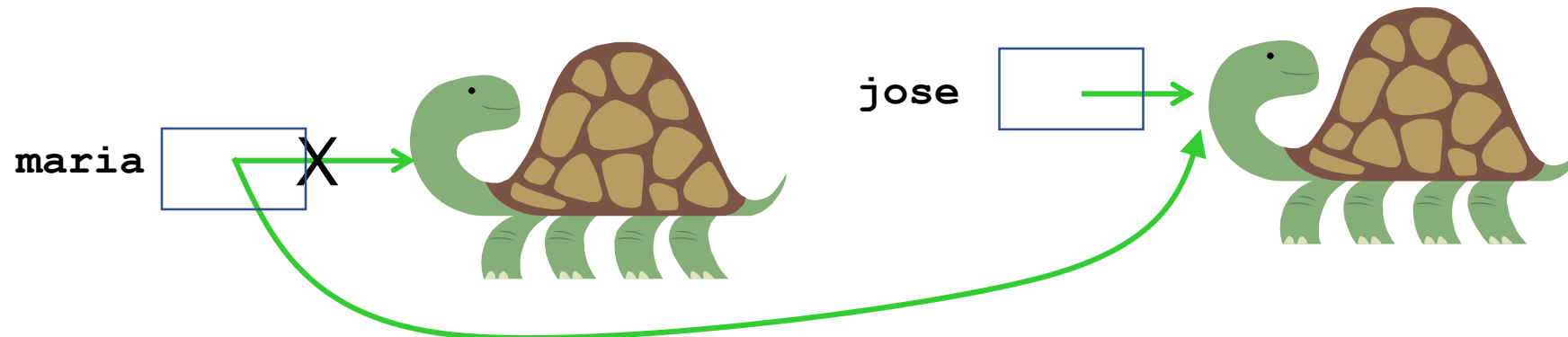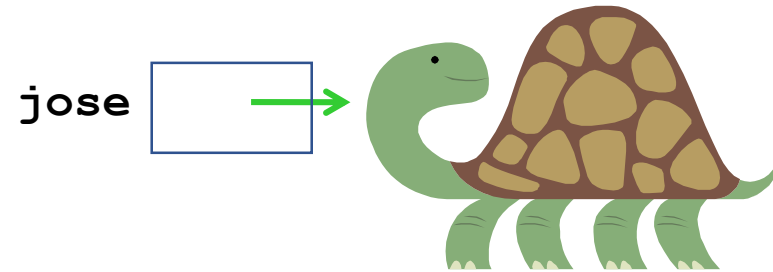
x [   ]   y [   ]

maria [   ]   jose [   ]

CS Concepts: References
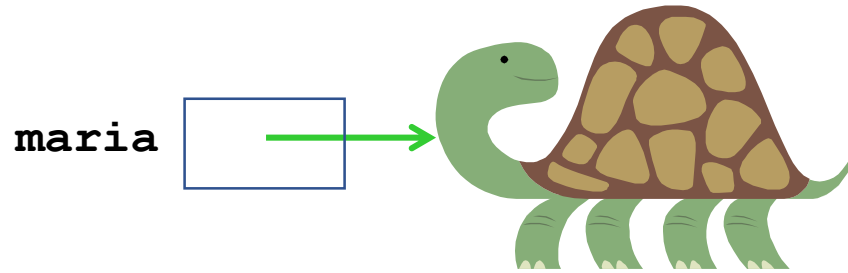
```
maria = turtle.Turtle()
jose = turtle.Turtle()
```


maria

jose

```
maria = jose
```


maria X

jose

CS Concepts: References

```
maria = turtle.Turtle()
jose = turtle.Turtle()
```

maria

jose

maria = jose

The arrows in maria and jose's boxes are just graphical representation of the reference (i.e., location of) the object in memory.

maria X

jose

CS Concepts: References

```
maria = turtle.Turtle()
jose = turtle.Turtle()
```

jose 352

maria 500

The arrows in maria and jose's boxes are just graphical representation of the reference (i.e., location of) the object in memory.
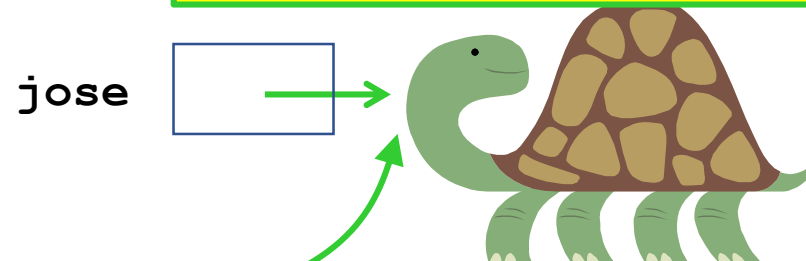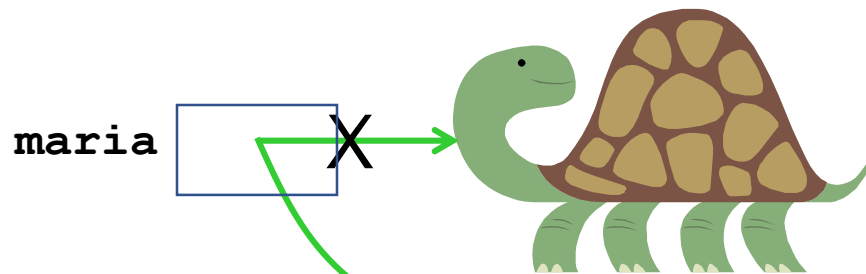
CS Concepts: References

```
maria = turtle.Turtle()
jose = turtle.Turtle()
```

maria

jose

The arrows in maria and jose's boxes are just graphical representation of the reference (i.e., location of) the object in memory.
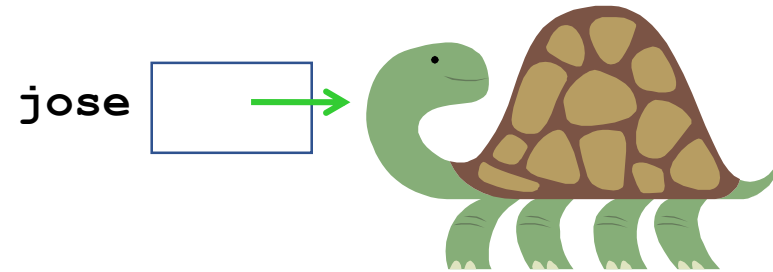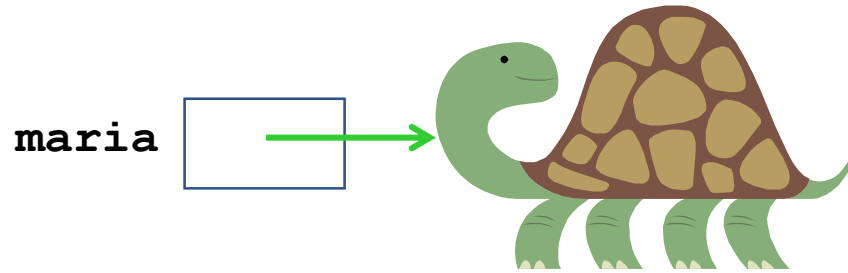
CS Concepts: References

```
maria = turtle.Turtle()
jose = turtle.Turtle()
```

jose   352

maria   500

maria = jose

The arrows in maria and jose's boxes are just graphical representation of the reference (i.e., location of) the object in memory.

352

maria   ~~500~~

jose   352

CS Concepts: References

```
maria = turtle.Turtle()
jose = turtle.Turtle()
```

jose 352

maria 500

maria = jose

The arrows in maria and jose's boxes are just graphical representation of the reference (i.e., location of) the object in memory.

jose 352

maria 352

CS Concepts: References

```
maria = turtle.Turtle()
jose = turtle.Turtle()
```

maria

jose

maria = jose

The arrows in maria and jose's boxes are just graphical representation of the reference (i.e., location of) the object in memory.
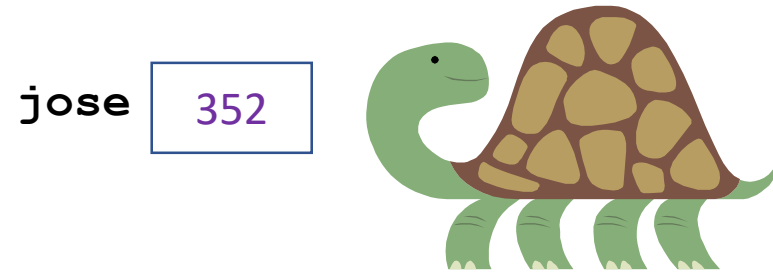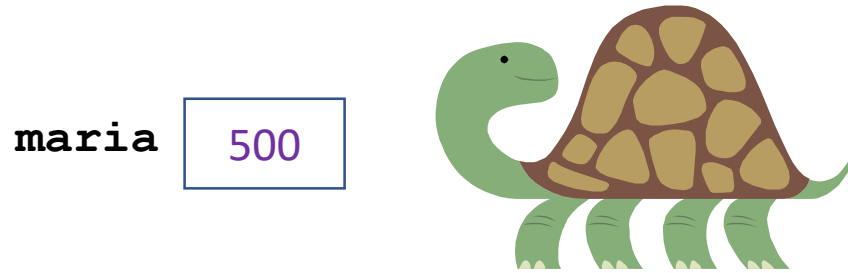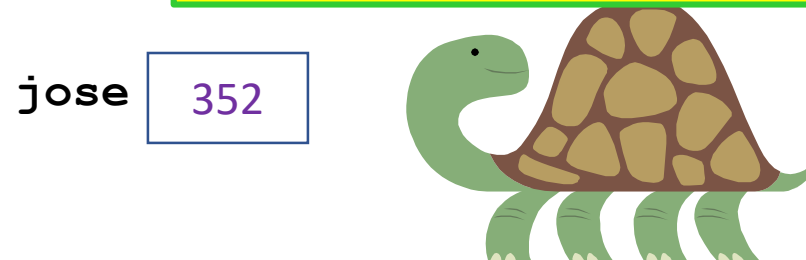
maria

jose

CS Concepts: References

```
maria = turtle.Turtle()
jose = turtle.Turtle()
```

The arrows in maria and jose's boxes are just graphical representation of the reference (i.e., location of) the object in memory.
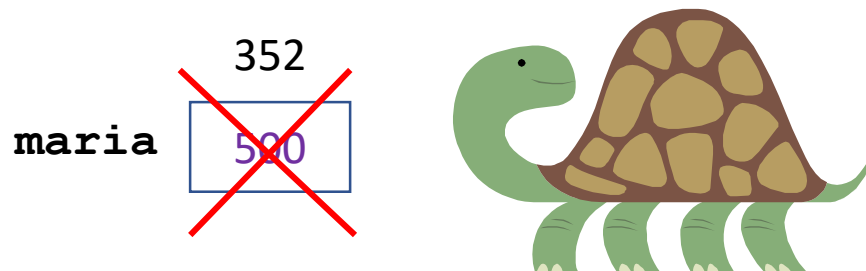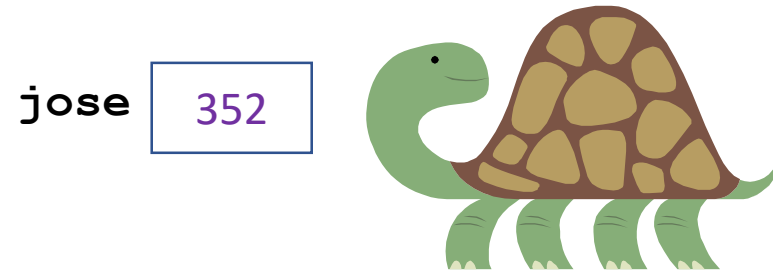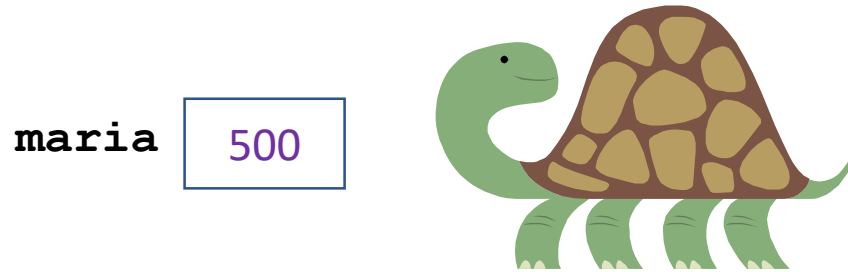
This holds for all Object types (not for primitive types)

THIS IS THE HARDEST THING YOU WILL LEARN IN CSE8A/CSE11. MASTER THIS IDEA AND YOU WILL ACE CSE8A/CSE11 (AND MUCH MORE)
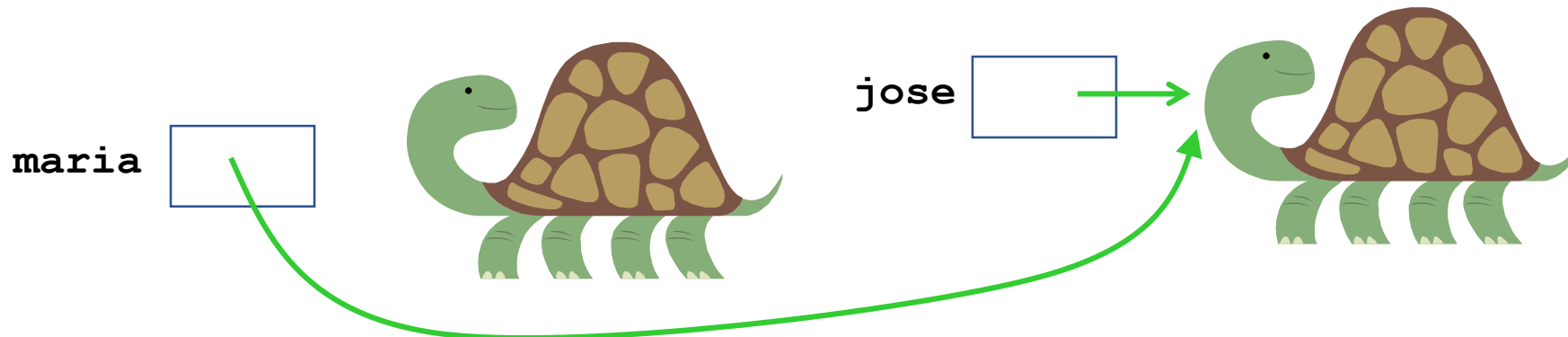
maria

jose

Corrected Memory Model

```
x = 42
y = 75
y = x
x = 101
```

Notice the difference!

x | 42 101   y | 75 42

Variable reassignment

```
maria = turtle.Turtle()
jose = turtle.Turtle()
maria = jose
jose.foward(100)
```

Object manipulation

maria | jose |

Variable assignment challenge

- What does this code draw?

```
import turtle
maria = turtle.Turtle()
jose = turtle.Turtle()
maria.penup()
jose.penup()
maria.setpos(-100, 0)
jose.setpos(-100, -50)
maria.pendown()
jose.pendown()
maria = jose
dist1 = 100
dist2 = 200
dist1 = dist2
dist2 += 150
maria.forward(dist1)
jose.forward(dist2)
```

Key ideas so far

- Variables have their own scope. When a function is called the *values* are passed in, not the variables themselves.

- Returning is not the same as printing. Only returning passes data back from a function.

- Conditional statements can be tricky! Practice, practice, practice!

- All variables in Python store references. References are memory addresses where data is located.

- Assignment statements copy the reference stored in the variable

- Two variables can store references to the same piece of data. If that piece of data can change (e.g. the Turtles) then the data seen by BOTH references will change.

Turtles and Functions

```
def drawShape(theTurtle):
    ''' Draw a simple shape with the turtle passed in '''
    theTurtle.forward(100)
    theTurtle.right(90)
    theTurtle.forward(100)
    theTurtle.right(90)
    theTurtle.forward(100)
    theTurtle.right(90)
    theTurtle.forward(100)
    theTurtle.right(90)
```

Introducing Lists
Lists allow you to store multiple values

```
def drawShapeWithLists(theTurtle):
    sideLengths = [100, 200, 50, 200]
    angles = [90, 120, 40, 60]
    theTurtle.forward(sideLengths[0])
    theTurtle.right(angles[0])
    theTurtle.forward(sideLengths[1])
    theTurtle.right(angles[1])
    theTurtle.forward(sideLengths[2])
    theTurtle.right(angles[2])
    theTurtle.forward(sideLengths[3])
    theTurtle.right(angles[3])
```

Introducing Tuples
Tuples *also* allow you to store multiple values (unordered)

```python
def drawShapeWithATuple(theTurtle):
    len_angle = (100, 90)
    theTurtle.forward(len_angle[0])
    theTurtle.right(len_angle[1])
    theTurtle.forward(len_angle[0])
    theTurtle.right(len_angle[1])
    theTurtle.forward(len_angle[0])
    theTurtle.right(len_angle[1])
    theTurtle.forward(len_angle[0])
    theTurtle.right(len_angle[1])
```

You can combine Lists and Tuples (or really, any types)

```python
def drawShapeWithListAndTuple(theTurtle):
    sides = [(100, 60), (200, 120), (100, 60), (200, 120)]
    side = sides[0]
    theTurtle.forward(side[0])
    theTurtle.right(side[1])
    side = sides[1]
    theTurtle.forward(side[0])
    theTurtle.right(side[1])
    side = sides[2]
    theTurtle.forward(side[0])
    theTurtle.right(side[1])
    side = sides[3]
    theTurtle.forward(side[0])
    theTurtle.right(side[1])
```

Indexing lists and tuples

```
sides = [(100, 60), (200, 120), (100, 60), (200, 120)]
value = sides[2]
```

What is the value of `value` after the assignment statement?

A. 60

B. 200

C. (200, 120)

D. (100, 60)

Indexing lists and tuples

```
sides = [(100, 60), (200, 120), (100, 60), (200, 120)]
value = sides[2]
```

Write as many Python statements (or sets of statements) as you can that will assign the value 120 to value, using the list sides

Key ideas so far

- Variables have their own scope. When a function is called the *values* are passed in, not the variables themselves.
- Returning is not the same as printing. Only returning passes data back from a function.
- Conditional statements can be tricky! Practice, practice, practice!
- All variables in Python store references. References are memory addresses where data is located.
- Assignment statements copy the reference stored in the variable
- Two variables can store references to the same piece of data. If that piece of data can change (e.g. the Turtles) then the data seen by BOTH references will change.
- Lists and tuples are compact ways of storing a "bunch" of data.
- You can access the individual elements in a list of a tuple using the index of that element. Indexes start at 0. The last index is the length of the list minus 1.

Not much is changing… is there an easier way to do this?

```python
def drawShapeWithListAndTuple(theTurtle):
    sides = [(100, 60), (200, 120), (100, 60), (200, 120)]
    side = sides[0]
    theTurtle.forward(side[0])
    theTurtle.right(side[1])
    side = sides[1]
    theTurtle.forward(side[0])
    theTurtle.right(side[1])
    side = sides[2]
    theTurtle.forward(side[0])
    theTurtle.right(side[1])
    side = sides[3]
    theTurtle.forward(side[0])
    theTurtle.right(side[1])
```

Not much is changing... is there an easier way to do this? YES! `for` loops!

```python
def drawShapeWithLoop(theTurtle):
    sides = [(100, 60), (200, 120), (100, 60), (200, 120)]


    for side in sides:

        theTurtle.forward(side[0])
        theTurtle.right(side[1])
```

Reading from a csv file

```python
import csv
…
hurricaneFile = "data/irma.csv"
# The line below is a little magical. It opens the file,
# with awareness of any errors that might occur.
with open(hurricaneFile, 'r') as csvfile:
    # This line gives you an "iterator" you can use to get each line
    # in the file.
    pointreader = csv.reader(csvfile)

    # You'll need to add some code here, before the loop

    for row in pointreader:
        print("Date:", row[0], "Time:", row[1])
```

Key ideas so far

- Variables have their own scope. When a function is called the *values* are passed in, not the variables themselves.

- Returning is not the same as printing. Only returning passes data back from a function.

- Conditional statements can be tricky! Practice, practice, practice!

- All variables in Python store references. References are memory addresses where data is located.

- Assignment statements copy the reference stored in the variable

- Two variables can store references to the same piece of data. If that piece of data can change (e.g. the Turtles) then the data seen by BOTH references will change.

- Lists and tuples are compact ways of storing a "bunch" of data.

- You can access the individual elements in a list of a tuple using the index of that element. Indexes start at 0. The last index is the length of the list minus 1.

- For loops allow you to repeat an action for each element in type of data that is "iteratable". Lists and tuples are iteratable. (The difference is tuples are not mutable—we'll get to that later).