Foundations of Programming

The Turtle, Lists and Tuples

Announcements

- These slides are on the FoCS website—have you found them?
- What is the role of your mentors?
- Partner expectations…
- Processes that might seem strange in SPIS are preparing you for fall

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

Review on parameters

```
# my own function!

def silly(x, y):
    """ returns x-y """
    return x-y
```

x [   ]

y [   ]

What is the value of `z` after the call to `silly`?

A. 19

B. -19

C. 2

D. 21

E. Nothing

```
>>> x = 21
>>> y = 2
>>> z = silly( y, x )
```

x [   ]

y [   ]

z [   ]

Review on parameters

```
# my own function!

def silly(x, y):
    """ does something silly """
    x = y-4
    return x-y
```

x [ ]

y [ ]

```
>>> x = 21
>>> y = 2
>>> z = silly(x, y)
>>> print(x, ",", y)
```

x [ ]

y [ ]

z [ ]

What is printed?

A. 21 , 2

B. 17 , 2

C. -2 , 2

D. -2 , -2

E. Something else

Flow of execution

```python
# my own function!

def greeting(personToGreet):
    """ prints a friendly greeting """
    print("Hello " + str(personToGreet))
    print("Welcome to SPIS!")
```

I can load this function by pressing F5.
I can then call it as follows:
```python
>>> name = "Christine"
>>> value = greeting("name")
```

What will be displayed after the last line in the code to the left?
A. Hello Christine
   Welcome to SPIS!
B. Hello name
   Welcome to SPIS!
C. Nothing
D. It will cause an error

Flow of execution

```python
# my own function!

def greeting(personToGreet):
    """ prints a friendly greeting """
    print("Hello " + str(personToGreet))
    print("Welcome to SPIS!")
```

I can load this function by pressing F5.
I can then call it as follows:
```
>>> name = "Christine"
>>> value = greeting("name")
Hello name
Welcome to SPIS!
>>> value
```

What will be displayed after the last line in the code to the left?
A. Welcome to SPIS!
B. Hello name
   Welcome to SPIS!
C. Nothing
D. It will cause an error

Flow of execution

```python
# my own function!

def greetingReturn(personToGreet):
    """ prints a friendly greeting """
    print("Hello " + str(personToGreet))
    return("Welcome to SPIS!")
```

I can load this function by pressing F5.

I can then call it as follows:

```
>>> name = "Christine"
>>> value = greetingReturn(name)
```

What will be displayed after the last line in the code to the left?
A. Welcome to SPIS!
B. Hello Christine
   Welcome to SPIS!
C. Nothing
D. It will cause an error

Flow of execution

```python
# my own function!

def greetingReturn(personToGreet):
    """ prints a friendly greeting """
    print("Hello " + str(personToGreet))
    return("Welcome to SPIS!")
```

I can load this function by pressing F5.
I can then call it as follows:
```
>>> name = "Christine"
>>> value = greetingReturn(name)
Hello Christine
>>> value
```

What will be displayed after the last line in the code to the left?
A. 'Welcome to SPIS!'
B. 'Hello Christine'
C. 'Hello Christine Welcome to SPIS!'
D. Nothing
E. It will cause an error

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.

More practice with Booleans and Conditionals

```
def isHurricane(windSpeed):
    ''' Determine if the windSpeed is strong enough to make it a hurricane'''
    if windSpeed >= 74:
        hurricane = True
        print "There is a hurricane!"
    else:
        hurricane = False
        print "Don't panic.  It's not a hurricane"
    return hurricane


wind = 74
hurricane = isHurricane(wind)
```

What gets printed by this code?
A. There is a hurricane!
   True
B. Don't panic. It's not a hurricane
   False
C. There is a hurricane!
D. Don't panic. It's not a hurricane
E. There is a hurricane!
   Don't panic. It's not a hurricane

More practice with Booleans and Conditionals

```python
def isHurricane(windSpeed):
    ''' Determine if the windSpeed is strong enough to make it a hurricane'''
    if windSpeed >= 74:
        hurricane = True
        print "There is a hurricane!"
    else:
        hurricane = False
        print "Don't panic.  It's not a hurricane"
    return hurricane


wind = 74
isAHurricane = isHurricane(wind)
```

What is the value of `isAHurricane`
A. True
B. False
C. Nothing
D. Something else

More practice with Booleans and Conditionals

```
def isHurricane(windSpeed):
    ''' Determine if the windSpeed is strong enough to make it a hurricane'''
    if windSpeed >= 74:
        hurricane = True
        print "There is a hurricane!"
    if windSpeed >= 157:
        hurricane = True
        print "It's a category 5!"
    else:
        hurricane = False
    return hurricane


wind = 100
isAHurricane = isHurricane(wind)
```

What is the value of `isAHurricane`
(Also, write what is printed)
A. True
B. False
C. Nothing
D. Something else

Compound booleans

```python
def isHurricane(windSpeed):
    ''' Determine if the windSpeed is strong enough to make it a hurricane'''
    if windSpeed >= 74 and windSpeed < 96:
        hurricane = True
        print "There is a category 1 hurricane"
    elif windSpeed >= 96 and windSpeed < 111:
        hurricane = True
        print "There is a category 2 hurricane"
    …


wind = 100
isAHurricane = isHurricane(wind)
```

More on `if` and `return`

```
def choice(x):
    if x > 10:
        print "A"
    if x > 5:
        print "B"
    if x > 0:
        print "C"
```

```
>>> ans = choice(8)
```

What will this function *return?*
A. "A"
B. "B"
C. "C"
D. More than one of these
E. None of these

More on `if` and `return`

```
def choice(x):
    if x > 10:
        print "A"
    if x > 5:
        print "B"
    if x > 0:
        print "C"
```

```
>>> ans = choice(8)
```

What will this function *print?*
A. "A"
B. "B"
C. "C"
D. More than one of these
E. None of these

More on `if` and `return`

```
def choice(x):
    if x > 10:
        print "A"
    elif x > 5:
        print "B"
    else x > 0:
        print "C"
```

```
>>> ans = choice(8)
```

What will this function *print?*
A. "A"
B. "B"
C. "C"
D. More than one of these
E. None of these

More on **if** and **return**

```
def choice(x):
    if x > 10:
        print "A"
    elif x > 5:
        print "B"
    else:
        print "C"
```

```
>>> ans = choice(8)
```

What will this function *print?*
A. "A"
B. "B"
C. "C"
D. More than one of these
E. None of these

More on `if` and `return`

```python
def choice(x):
    if x > 10:
        return "A"
    if x > 5:
        return "B"
    if x > 0:
        return "C"
```

```
>>> ans = choice(8)
```

What will this function *print?*
A. "A"
B. "B"
C. "C"
D. More than one of these
E. None of these

What if x is <= 0?

More on `if` and `return`

```python
def choice(x):
    if x > 10:
        return "A"
    if x > 5:
        return "B"
    if x > 0:
        return "C"
```

```
>>> ans = choice(8)
```

What will this function *return?*
A. "A"
B. "B"
C. "C"
D. More than one of these
E. None of these

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!

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)
```

firstTurtle

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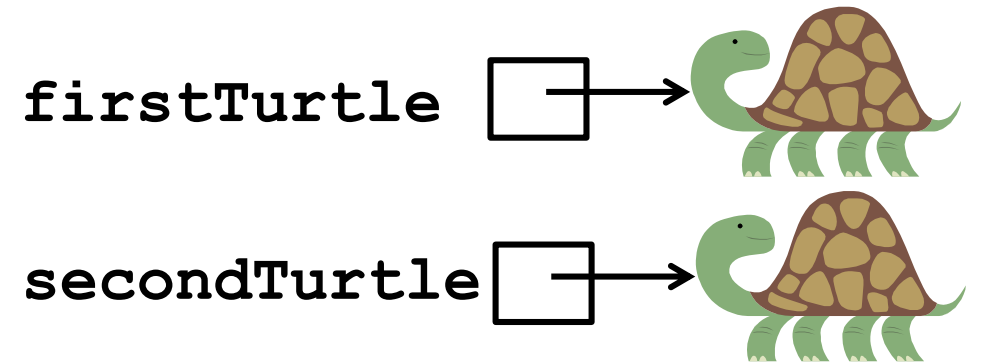
Two different turtle "objects" with the same capabilities
The variables store "references" (location of the Turtle in memory) to each Turtle.

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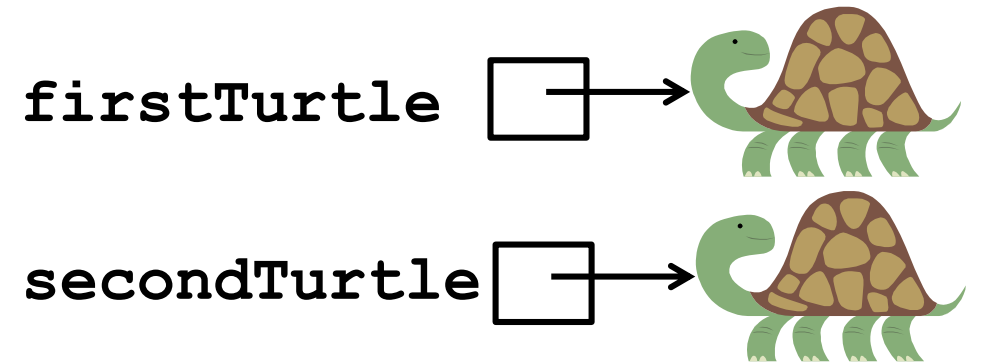
How does the diagram change with the line in red? How does that change what is drawn?

PSA 1: Your first program

```
public class DrawMyName
{
    // The line below is magic, you don't have to understand it (yet)
    public static void main (String[] args)
    {
        // This is a comment
        World w = new World();

        //Creates a turtle in w at (x,y) (200,100)
        Turtle jose = new Turtle(200,100,w);

        //Make a U shape
        jose.forward(55);
        jose.turn(90);
        jose.forward(30);
        jose.turn(90);
        jose.forward(45);
        jose.forward(10);
        jose.turn(90);

    }
}
```

Your program starts here

Each line is executed in order

Don't worry about details (yet).
The big picture is:
- jose is a Turtle
- He can move around the screen responding to specific commands in order
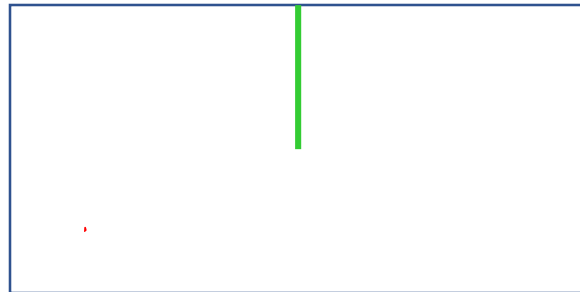- You arrange those commands to make him draw what you want him to

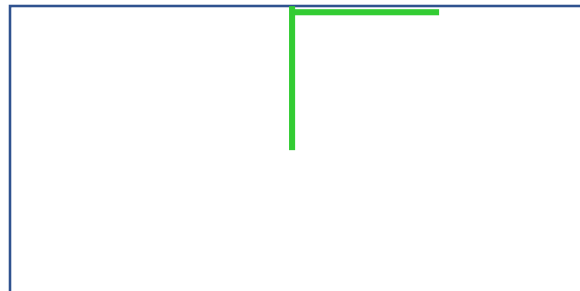# CS Concept: References

• What does this code draw?

```
World world1 = new World(200,100);
Turtle maria = new Turtle(25, 25, world1);
Turtle jose = new Turtle(100, 50, world1);
maria = jose;
maria.forward(50);
jose.turn(90);
jose.forward(50);
```

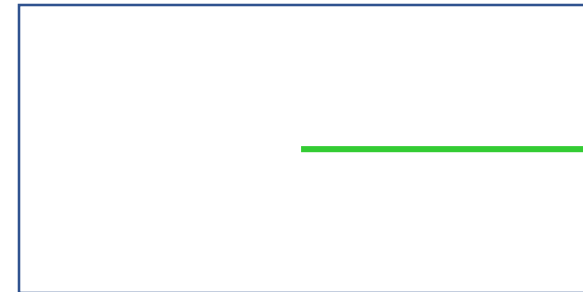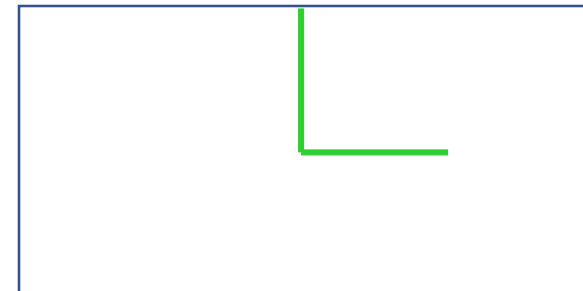Y doesn't change? So why does maria's turtle move when you move jose??

A

B

C

D

```
>> int x = 42;
>> int y = x;
>> x = 101;
>> S.o.p(x);
101
>> S.o.p(y);
42
```

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);
```
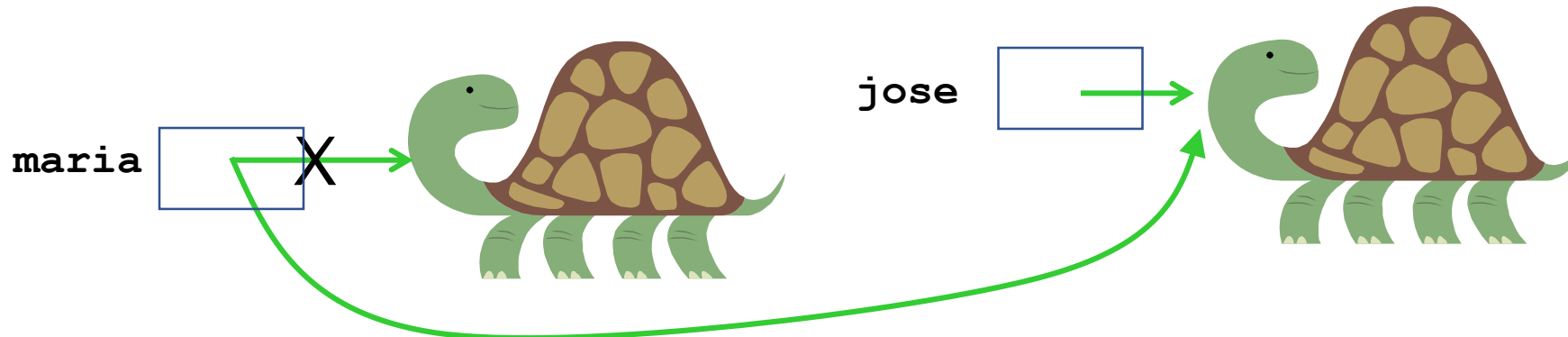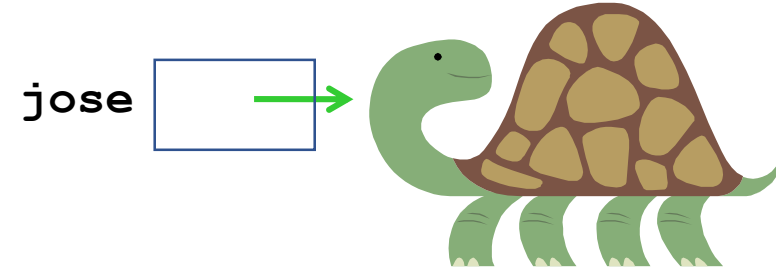
x [ ]   y [ ]

maria [ ]   jose [ ]

CS Concepts: References

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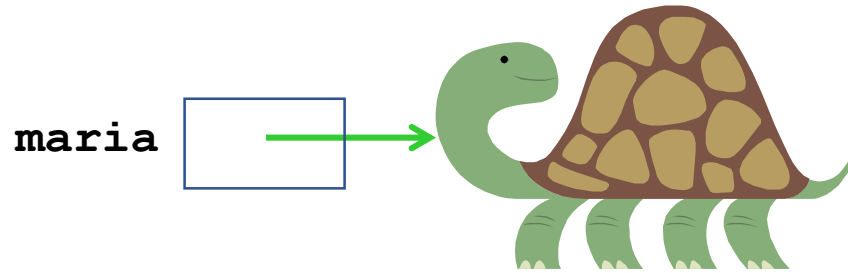
maria

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.
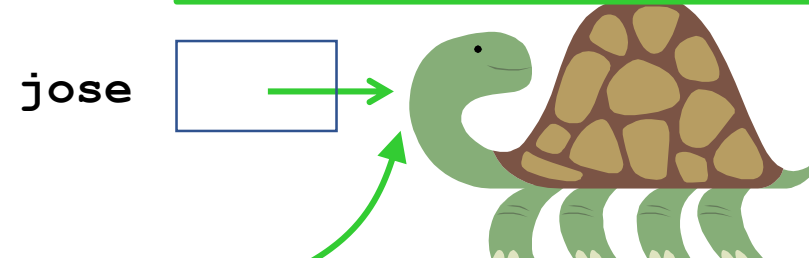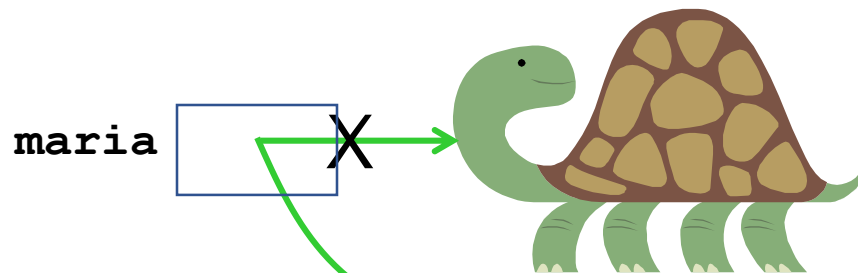
maria

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.
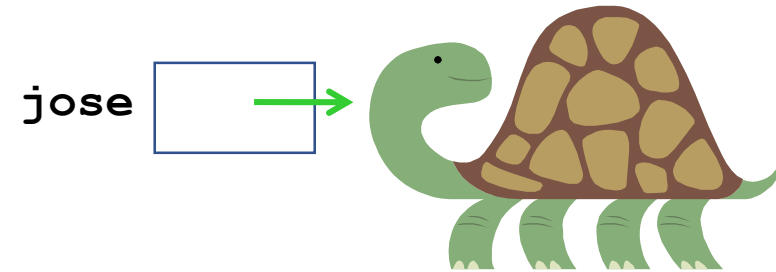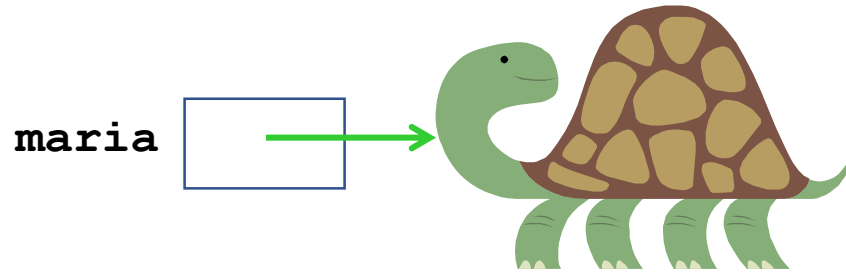
CS Concepts: References

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

maria `500`

jose `352`

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.

maria 352

jose 352

CS Concepts: References

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


maria

jose

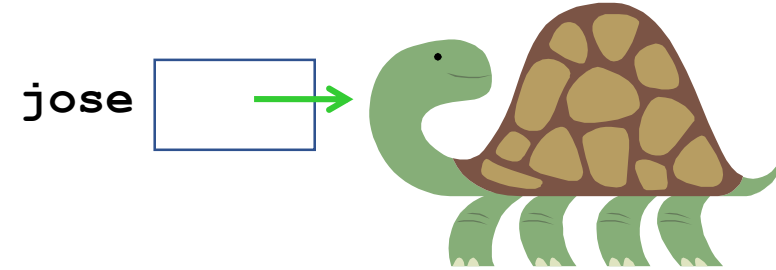**maria = jose**

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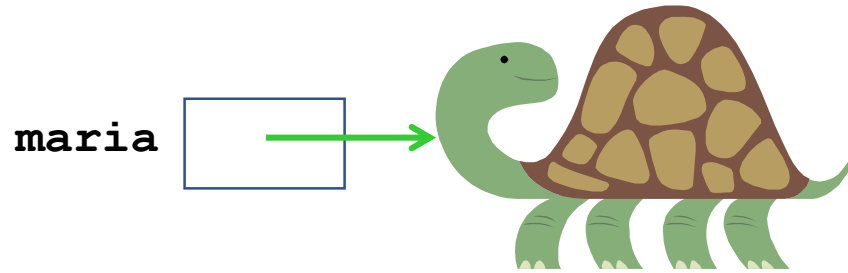
The arrows in maria and jose's boxes are just graphical representation of the reference (i.e., location of) the object in memory.

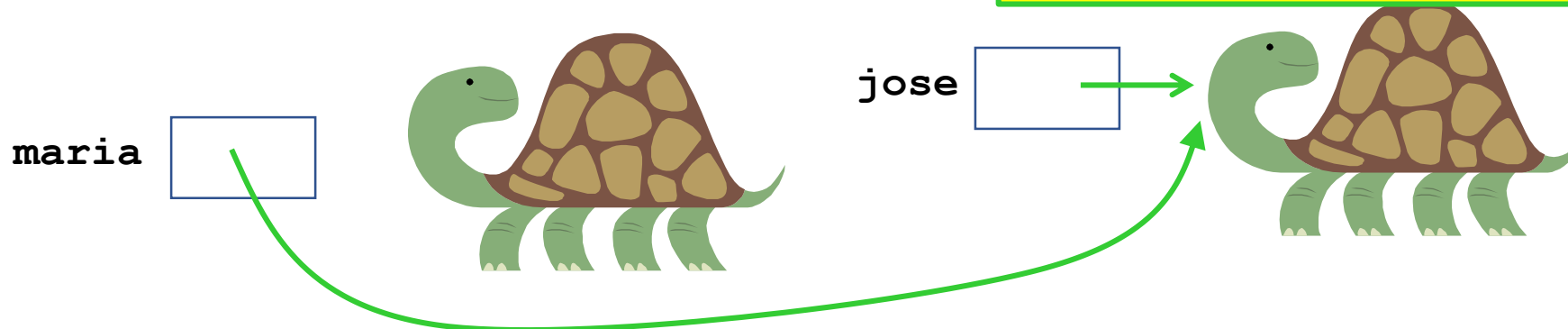This holds for all Object types (not for primitive types)

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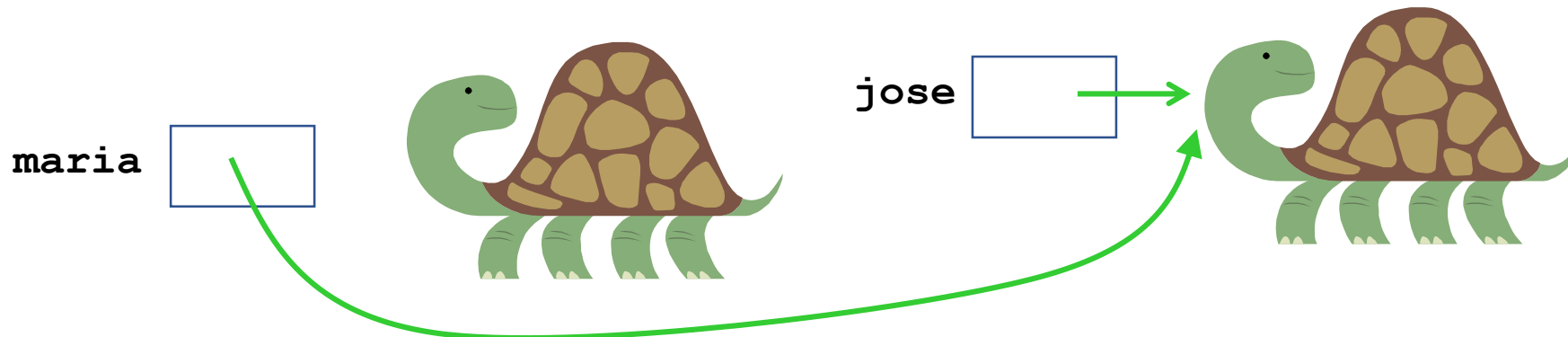
jose

maria

Corrected Memory Model

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

Notice the difference!

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

Object manipulation

x | 42 101   y | 75 42

Variable reassignment
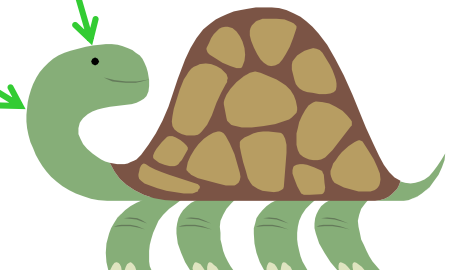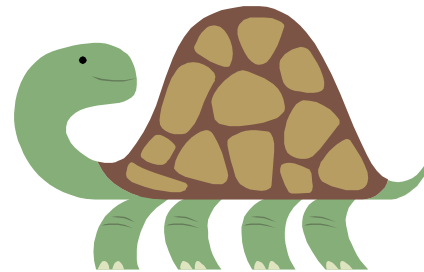
maria | | jose | |

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

```python
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)

```
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:
        # This code just prints out each row in the file
        # You'll need to change it
        for data in row:
            print(data, ' ', end='')
        print()
```

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).**