Getting Into the Details

```
def change(inside_number, inside_list):
  inside_number = 5
  inside_list[0] = 7
  return inside_number
outside_number = 12
outside_list = [1,2,3]
result = change(outside_number, outside_list)
```

- What are the values at the end of `outside_number`, `outside_list` and `result`?

Memory Management

- Quick review:
  - Computers store all data in memory
  - All data is stored in binary form
  - The type of the data (e.g. int, string, image) is used by functions to know how to process the data
    - 109 can be the number 109, the character 'm', a color value, etc.

Memory Management

- Assigning a new variable allocates memory for its value

```
a = 19
```

  - Creates a new integer object with the value 19
  - Makes the name (symbol) a reference that object

```
a = [1,2,3,4]
```

  - Creates a new list object with 4 slots
  - Creates 4 new integer objects with values 1,2,3,4
  - Makes each slot in the list reference an integer object
  - Makes the name (symbol) a reference the list object

Heap and Scope

- All objects are allocated on a *heap*, a chunk of memory reserved for the program
- Variables exist in the *scope* of the module or function that they are created in
  - A variable name cannot be used outside its scope; that's a different variable
    - E.g. Portland, OR vs. Portland, Maine

Assignment

- Assigning a variable
  - Creates that variable if it doesn't exist
  - Makes it reference the value given
- Assigning a variable to another variable
  - `a = 17`
  - `b = a`
  - Makes the symbol `b` reference the same object that `a` references

Assignment

- Assigning a variable
  - Creates that variable if it doesn't exist
  - Makes it reference the value given
- Assigning a variable to another variable

  `a = 17`
  - a references an integer object with value 17

  `b = a`
  - b references the *same* object

  `a = 18`
  - a references a *different* integer object with value 18
  - (b still references 17)

Assignment

- Assigning a variable to another variable

```
a = [17,18]
```

  - `a` references a list object referencing objects for 17 and 18

```
b = a
```

  - `b` references the *same* list object

```
b[0] = 16
```

  - The shared list now references object with values 16 and 18
    - i.e. `a` and `b` both reference `[16,18]`

```
a = [20,21]
```

  - `a` references a new list object
  - `b` still references the old list with `[16,18]`

Getting Into the Details

```
def change(inside_number, inside_list):
  inside_number = 5
  inside_list[0] = 7
  return inside_number
outside_number = 12
outside_list = [1,2,3]
result = change(outside_number, outside_list)
```

- Values at the end:
  - `outside_number => 12`
  - `outside_list => [7,2,3]`
  - `result => 5`

Getting Into the Details

- Passing parameter values into a function works exactly the same way that assigning a variable to another variable works!
  - The parameter inside the function is assigned the value of the variable (or literal) that is passed in