Review Questions

8.

I think it's easiest to show this with a little code. Say you have the following:

```
def square(x):
  return x**2

print(square(3))
```

In the above code, the `x` in `def square(x)` is considered a formal argument. Formal arguments are identifiers in the prototype for methods, and are a "placeholder" value. On the other hand, the `3` in `print(square(3))` is an actual parameter. Actual arguments are the actual value passed into the method.

21.

Ruby (and python) uses pass-by-assignment parameter passing. This is because all data values are objects in these languages. It's called pass-by-assignment because the actual arguments are assigned to the formal parameters. This may seem close to pass-by-reference, but they're actually quite different. This is because most objects in these languages are immutable and so essentially act as though they were being passed-by-value.

Problem Set

5.

void swap(int a, int b) { int temp; temp = a; a = b; b = temp; } void main() { int value = 2, list[5] = {1, 3, 5, 7, 9}; swap(value, list[0]); swap(list[0], list[1]); swap(value, list[value]); }

Pass by Value

In pass by value, nothing will change. So `value = 2, list[5] = {`1, 3, 5, 7, 9`}` will remain true for all of the calls.

Pass by Reference

`swap(value, list[0]);` `value = 1, list[5] = {2, 3, 5, 7, 9}`

`swap(list[0], list[1]);` `value = 1, list[5] = {3, 2, 5, 7, 9}`

`swap(value, list[value])` `value = 2, list[5] = {3, 1, 5, 7, 9}`

Pass by value-result

`swap(value, list[0]);` `value = 1, list[5] = {2, 3, 5, 7, 9}`

`swap(list[0], list[1]);` `value = 1, list[5] = {3, 2, 5, 7, 9}`

`swap(value, list[value])` `value = 2, list[5] = {3, 1, 5, 7, 9}`