Python 103
Understanding Python's Memory Model & Mutability

+Wesley Chun, Principal
CyberWeb Consulting
wescpy@gmail.com :: @wescpy
cyberwebconsulting.com
I Write

[Image of various Python-related publications]

I Teach

[Image of various logos from tech companies and conferences]
About this talk & you

• Agenda
  • Review of Objects, References, Data Types
  • Mutability & memory referencing gotchas
  • But wait! There's more... maybe (time-permitting)
    • Based on 90-minute EuroPython 2011 talk
  • Content plagiarized from Python course and Core Python Programming, http://amzn.com/0132269937

• About you
  • Some Python experience... not (nec) a beginner
  • Still don't understand “weird” behavior
    • Bugs but you swear code is correct
  • Want to learn more internals

Python Objects

• Objects are primary abstraction for data

• Attributes of all objects
  • Identity (similar to a memory address)
  • Type
  • Value

• All attributes are read-only except perhaps value

• Python has 30+ object types in all
  • docs.python.org/ref/types.html

(c) 1998-2013 CyberWeb Consulting.
All rights reserved.
Object Types

• Standard Types
  • Numbers (3-8)
  • Strings (2-3)
  • Lists
  • Tuples
  • Dictionaries
  • Sets (2)

• Some Other Types
  • None
  • Files
  • Functions/Methods
  • Modules
  • Types/Classes
  • Iterators/Generators

Objects and References

• Objects allocated on assignment
• All objects passed by reference
  • References are also called *aliases*
  • *Reference count* used to track total number
  • Count in/decrements based upon usage
  • Objects garbage-collected when count goes to 0
More on References

• Variables don't "hold" data per se (not memory)
• Variables just point to objects (aliases)
• Additional aliases to an object can be created
• Objects reclaimed when "refcount" goes to 0
• Be aware of cyclic references

\[
x = 1 \\
y = x
\]

Reference Count Increased

• Examples of refcount increment:
  • It (the object) is created (and assigned)
    \[
    \text{foo} = 'Python is cool!'
    \]
  • Additional aliases for it are created
    \[
    \text{bar} = \text{foo}
    \]
  • It is passed to a function (new local reference)
    \[
    \text{spam}(\text{foo})
    \]
  • It becomes part of a container object
    \[
    \text{lotsaFoos} = [123, \text{foo}, 'xyz']
    \]
Reference Count Decreased

• Examples of refcount decrement:
  • A local reference goes out-of-scope
    i.e., when spam() ends
  • Aliases for that object are explicitly destroyed
    del bar  # or del foo
  • An alias is reassigned a different object
    bar = 42
  • It is removed from a container object
    lotsaFoos.remove(foo)
  • The container itself is deallocated
    del lotsaFoos  # or out-of-scope

Categorizing the Standard Types

• Why?
  • To make you learn them faster
  • To make you understand them better
  • To know how to view them internally
  • To encourage more proficient programming

• Three Models
  • Storage
  • Update
  • Access
### Storage Model

- How data is stored in an object
- Can it hold single or multiple objects?

<table>
<thead>
<tr>
<th>Model Category</th>
<th>Python Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>literal/scalar</td>
<td>numbers (all), strings</td>
</tr>
<tr>
<td>container</td>
<td>lists, tuples, dicts, sets</td>
</tr>
</tbody>
</table>

### Update Model

- Can an object's value be updated?
- Mutable == yes and immutable == no
- There is one of each set type
- `bytearray` type is mutable (3.x)

<table>
<thead>
<tr>
<th>Model Category</th>
<th>Python Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>mutable</td>
<td>lists, dicts, sets</td>
</tr>
<tr>
<td>immutable</td>
<td>numbers, strings, tuples, frozensets</td>
</tr>
</tbody>
</table>
### Access Model

- How data is accessed in an object
- Directly, via index, or by key
- Primary model for type differentiation

<table>
<thead>
<tr>
<th>Model Category</th>
<th>Python Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>direct</td>
<td>numbers, sets</td>
</tr>
<tr>
<td>sequence</td>
<td>strings, lists, tuples</td>
</tr>
<tr>
<td>mapping</td>
<td>dicts</td>
</tr>
</tbody>
</table>

### Type Categorization Summary

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Storage Model</th>
<th>Update Model</th>
<th>Access Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>numbers</td>
<td>literal/scalar</td>
<td>immutable</td>
<td>direct</td>
</tr>
<tr>
<td>strings</td>
<td>literal/scalar</td>
<td>immutable</td>
<td>sequence</td>
</tr>
<tr>
<td>lists</td>
<td>container</td>
<td>mutable</td>
<td>sequence</td>
</tr>
<tr>
<td>tuples</td>
<td>container</td>
<td>immutable</td>
<td>sequence</td>
</tr>
<tr>
<td>dictionaries</td>
<td>container</td>
<td>mutable</td>
<td>mapping</td>
</tr>
<tr>
<td>sets</td>
<td>container</td>
<td>im/mutable</td>
<td>direct</td>
</tr>
</tbody>
</table>
Objects & References Quiz

• What is the output of the code below? WHY?

Example 1

```python
x = 42
y = x
x = x + 1
print x
print y
```

Example 2

```python
x = [1, 2, 3]
y = x
x[0] = 4
print x
print y
```

Quiz Analysis

• Example 1

```python
x = 42
y = x
x += 1
```

• Example 2

```python
x = [1, 2, 3]
y = x
x[0] = 4
```
## Quiz Answers

### Example 1

```python
>>> x = 42
>>> y = x
>>> x += 1
>>> print x
43
>>> print y
42
```

### Example 2

```python
>>> x = [1, 2, 3]
>>> y = x
>>> x[0] = 4
>>> print x
[4, 2, 3]
>>> print y
[4, 2, 3]
```

## Quiz Epilogue

- **Example 1**
  
  ```
  x = 42
  y = x
  x += 1
  ```

- **Example 2**
  
  ```
  x = [1,2,3]
  y = x
  x[0] = 4
  ```

(c) 1998-2013 CyberWeb Consulting.  
All rights reserved.
Digging deeper into Python

- We know what this does...

  \[
  \begin{align*}
  x &= 4 \\
  y &= x
  \end{align*}
  \]

- What about this?

  \[
  \begin{align*}
  x &= 4 \\
  y &= 4
  \end{align*}
  \]

Interning of Objects

- Exception to the general rule
- Some strings and integers are "interner"ed
  - Integers in `range(-5, 257)` [currently]
  - Oft-used, single-character, and empty strings
- Primarily for performance reasons only

  \[
  \begin{align*}
  x &= 4 \\
  y &= 4
  \end{align*}
  \]

  \[
  \begin{align*}
  x &= 4.3 \\
  y &= 4.3
  \end{align*}
  \]
Objects and References Quiz 2

- Copy objects using their factory function(s)
- Can use improper slice with sequences
- What is the output here (and WHY)?

```python
x = ['foo', [1, 2, 3], 10.4]
y = list(x) # or x[:]
y[1][0] = 4
print x
print y
```

Quiz analysis

- Copying a list

```python
x = ['foo', [1, 2, 3], 10.4]
y = list(x) # or x[:]
y[1][0] = 4
```

```
x ['foo', [1, 2, 3], 10.4]
y ['foo', [1, 2, 3], 10.4]
```
Quiz 2 Answer

```python
>>> x = ['foo', [1,2,3], 10.4]
>>> y = list(x) # or x[:]
>>> y[1][0] = 4
>>> print x
['foo', [4, 2, 3], 10.4]
>>> print y
['foo', [4, 2, 3], 10.4]
```

Correct analysis

- Copying a list means copying references, not...

```
x = ['foo', [1, 2, 3], 10.4]
y = list(x) # or x[:]
y[1][0] = 4
```

(c) 1998-2013 CyberWeb Consulting.
All rights reserved.
Copying Objects

• Trickier with mutable objects

• Let's say you have a list a you wish to copy to b

  • Creating an alias not a copy
    \[ b = a \quad \# \quad a == b \quad \text{and} \quad a \text{ is } b \quad [id(a) == id(b)] \]

  • Creating a shallow copy (all objects inside are aliases!)
    \[ b = a[:] \quad \# \quad a == b \quad \text{but} \quad a \text{ is not } b \]

  • Creating a deep copy (all objects inside are copies)

  • Use the `deepcopy()` function in the `copy` module
    \[ b = \text{copy}.deepcopy(a) \]

Function Evaluation

• "Call by reference" or "call by value"? Neither. Both.

• Behavior is based on whether object is mutable

  • Recall that objects are always passed by reference

  • For mutable objects, aliases behave like pointers... a change in one changes all, like call by reference

  • BUT for immutable objects, acts like call by value
Summary

1. Know about objects, references, and aliases
2. Know difference between all standard data types
3. Mutability could be causing your bug
4. Diagram deeper issues... may shed light
5. Use iterators and avoid unnecessary lists
   (generator expressions instead of list comprehensions)

uids = set([int(uid) for uid in data.split(',')])