Mutability, Nonlocal, Exceptions

10/14/16
What is mutation?

- Mutation is the changing of values.
- Certain data types in Python are mutable:
  - Lists
- Other data types in Python are immutable:
  - Tuples
  - Strings
- Dictionary keys must be immutable.
- Dictionary values can be mutable or immutable.
Brief Intro to Mutability in HW 4

- Instead of returning a new list, we could have modified `lst`, which would be an example of mutation.

```python
def replace_elem(lst, index, elem):
    """Create and return a new list whose elements are the same as those in LST except at index INDEX, which should contain element ELEM instead."

>>> old = [1, 2, 3, 4, 5, 6, 7]
>>> new = replace_elem(old, 2, 8)
>>> new
[1, 2, 8, 4, 5, 6, 7]
>>> new is old  # check that replace_elem outputs a new list
False
"""
```
Examples of Immutable Data Types

```python
>>> x = (1, 2, 3)
>>> x[0] = 10  # What will this do?

>>> d = {}
>>> key = [1, 2]
>>> value = [3, 4]
>>> d[key] = value  # What about this?
```
Mutability is Tricky

- Mutability can often lead to unexpected behavior when writing programs
- [http://tinyurl.com/zexl7he](http://tinyurl.com/zexl7he)

- Both variables refer to the same list in the above example
- It’s easy to mistake x and y as being two different lists
Examples of Mutable Data Types

- List creation: http://tinyurl.com/j4jc5gg
- Appending to a list: http://tinyurl.com/jnteyar
- Nested lists: http://tinyurl.com/j57szgu

- These sorts of scenarios can often lead to buggy code
- Understanding the basics of mutability really helps in debugging your code
**`is` vs. `==`**

- `==` only compares values, “is” compares whether two variables actually point to the same list

```python
>>> x = [1, 2, 3, 4]
>>> y1 = x
>>> y2 = list(x)
>>> y1 is x
True
>>> y2 is x
False
>>> y1 == x
True
>>> y2 == x
True
```
We're almost done grading midterms.
Mid-semester survey to come out soon. We would really appreciate everyone’s feedback!
Prof. Friedland will not be having office hours this and next week. He is still reachable by email.
Clarification on slip day/late policy

Any questions?
Mutability and Nonlocal

- Consider the following example:

```python
def outer():
    x = 5
    def inner():
        x = 6  # Will this change the value of the outer x?
        return inner()
```
def outer():
    x = 5
    def inner():
        x = 6  # Will this change the value of the outer x?
        return inner()

    inner() does not modify the outer variable; it will create a new local variable

    http://tinyurl.com/jxxanzl

    However.... http://tinyurl.com/jluwmfg
Mutation and Nonlocal

- Mutable values can be changed inside inner()
- To change immutable values inside inner(), we must use the nonlocal keyword
- [http://tinyurl.com/j42yu3w](http://tinyurl.com/j42yu3w)

- Nonlocal will not allow you to change global variables in this manner
- To do this, you must use the global keyword [http://tinyurl.com/z766886](http://tinyurl.com/z766886)
Exceptions

- Python raises an exception whenever an error occurs
  - ZeroDivisionError
  - ValueError

- Exceptions can be handled by the program, preventing a crash (next slide)
- Programs can also raise exceptions of their own (later in the course)
Handling Exceptions

- The following function won’t cause the program to crash, even if you try to divide by 0

```python
def safe_divide(x, y):
    quotient = "Error"
    try:
        quotient = x/y
    except ZeroDivisionError:
        print("Can’t divide by zero!")
    return quotient
```