T2 — at end of lecture
A2 — in sequence...

CSC148 fall 2013

identity, information hiding, reduce
week 11

Danny Heap
heap@cs.toronto.edu
BA4270 (behind elevators)
http://www.cdf.toronto.edu/~heap/148/F13/
416-978-5899

November 20, 2013

Computer Science
UNIVERSITY OF TORONTO

Outline

hiding attributes

equality

reduce

why not hidden by default?

The Python approach is to start with easy-to-access attributes, and only worry about restricting them when it becomes an issue:

```
class RegexTreeNode:
    """A Regex Tree node"""
    def __init__(self: 'RegexTreeNode', symbol: str,
                 children: list) -> None:
        """A new RegexTreeNode with regex symbol and subtrees children.
        REQ: symbol must be one of "0", "1", "e", "|", ".", "*"

        >>> print(RegexTreeNode("0", []))
        RegexTreeNode('0', [])
        >>> print(RegexTreeNode("1", []))
        RegexTreeNode('1', [])
        """
        self.symbol = symbol
        self.children = children[:]
```

easy-to-use versus managed attributes

Python attributes are easy to use, but over time we may refine the implementation. Without changing the interface, how can you change the data structure representing an attribute, or use some computation in getting or setting it?

For example, what if we wanted to **enforce** `RegexNode` `symbol` being one of '1', '0', 'e', '.', '—', or '*'?

properties

built-in function property gives us nuanced access:

```
def get_symbol(self: 'RegexTreeNode') -> object:
    """get private symbol."""
    return self._symbol

def set_symbol(self: 'RegexTreeNode', s: str) -> None:
    """set private symbol"""
    if not s in '01e|.*':
        raise Exception('Invalid symbol: {}'.format(s))
    else:
        self._symbol = s
```

get_symbol, set_symbol

```
symbol = property(get_symbol, set_symbol, None, None)
```

read-only

We could also make `RegexTreeNode` `children` read-only:

```
def get_children(self: 'RegexTreeNode') -> list:
    """get private children"""
    return self._children

children = property(get_children, None, None, None)
```

when are objects equal?

The default behaviour of == is to report whether two objects are the same. . . object!

Sometimes we want to know whether they are **equivalent**, and we have various notions of equivalence.

Customize with `__eq__`

equivalent RegexTreeNodes

We really want to check whether they have the same symbol and the same children:

```
def __eq__(self: 'RegexTreeNode', other: 'RegexTreeNode') -> bool:
    """is this RTN equivalent to other?"""
    return (self.symbol == other.symbol and
            all([c1.__eq__(c2)
                 for c1, c2 in zip(self.children,other.children)]))
```

bundle up an iterable

Google has said they could never spread operations over many servers without MapReduce

map

You already have the idea of `map` — it's very similar to list comprehensions, although Python also has `map`

[<exp of i> for i in <iter>]

parallelize computation, some expression on separate CPUs

Reduce allows you to combine the elements of an iterable into a single, somehow reduced, value.

long multiplication?

Suppose you want to multiply all the numbers in a list:

```
from functools import reduce
reduce(int.__mul__, [1, 2, 3, 4, 5])
```

what functions can reduce?

Functions that take two arguments of the same type, and return a value of the same type. Reduce the iterable, pairwise, to one value.

```
def my_add(x, y):
    return x + y

def my_sum(L: 'iterable of addables...') -> object:
    return reduce(my_add, L, 0)

# (((((0 + L[0]) + L[1]) + L[2]) + L[3]) + L[4]) ...
```

special cases: **sum, max, min**