6.009
Fundamentals of Programming

Week 4 Lecture:
Recursion

Adam Hartz
hz@mit.edu

What is Recursion?

In a general sense, *recursion* occurs when a thing is defined in terms of itself.

Example: For nonnegative integer n,

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \times (n-1)! & \text{otherwise} \end{cases}$$

To solve a problem recursively, we typically identify:

- One of more simple **base cases** (a terminating scenario that does not use recursion to produce an answer), and
- A **recursive case** (a set of rules that reduce all other cases toward the base case).

Example: Factorial

```
def factorial(n):
    if n == 0:
        return 1
    return n * factorial(n - 1)
```

What happens when we call `factorial(0)`?
`factorial(2)`?

Recursion vs Iteration?

Factorials can also be computed iteratively.

```
def factorial(n):
    if n == 0:
        return 1
    return n * factorial(n - 1)
```

```
def factorial(n):
    out = 1
    for i in range(1, n+1):
        out *= i
    return out
```

Which would you choose? Why?

Recursion vs Iteration?

Factorials can also be computed iteratively.

```
def factorial(n):
    if n == 0:
        return 1
    return n * factorial(n - 1)
```

```
def factorial(n):
    out = 1
    for i in range(1, n+1):
        out *= i
    return out
```

Which would you choose? Why?

Do we even need recursion? Can't we always solve the problem iteratively?

Recursion In The Wild

Example from CAT-SOOP:

```
def can_log(x):
    """
    Checks whether a given value can be a log entry.

    Valid log entries are strings/bytestrings, ints, floats, complex numbers,
    None, or Booleans; _or_ lists, tuples, sets, frozensets, dicts, or
    OrderedDicts containing only valid log entries.
    """
```

Recursion In The Wild

Example from CAT-SOOP:

```
def can_log(x):
    """
    Checks whether a given value can be a log entry.

    Valid log entries are strings/bytestrings, ints, floats, complex numbers,
    None, or Booleans; _or_ lists, tuples, sets, frozensets, dicts, or
    OrderedDicts containing only valid log entries.
    """
    if isinstance(x, (str, bytes, int, float, complex, NoneType, bool)):
        return True
```

Recursion In The Wild

Example from CAT-SOOP:

```
def can_log(x):
    """
    Checks whether a given value can be a log entry.

    Valid log entries are strings/bytestrings, ints, floats, complex numbers,
    None, or Booleans; _or_ lists, tuples, sets, frozensets, dicts, or
    OrderedDicts containing only valid log entries.
    """
    if isinstance(x, (str, bytes, int, float, complex, NoneType, bool)):
        return True
    elif isinstance(x, (list, tuple, set, frozenset)):
        return all(can_log(i) for i in x)
    elif isinstance(x, (dict, OrderedDict)):
        return all((can_log(k) and can_log(v)) for k,v in x.items())
```

Recursion In The Wild

Example from CAT-SOOP:

```
def can_log(x):
    """
    Checks whether a given value can be a log entry.

    Valid log entries are strings/bytestrings, ints, floats, complex numbers,
    None, or Booleans; _or_ lists, tuples, sets, frozensets, dicts, or
    OrderedDicts containing only valid log entries.
    """
    if isinstance(x, (str, bytes, int, float, complex, NoneType, bool)):
        return True
    elif isinstance(x, (list, tuple, set, frozenset)):
        return all(can_log(i) for i in x)
    elif isinstance(x, (dict, OrderedDict)):
        return all((can_log(k) and can_log(v)) for k,v in x.items())
    return False
```

Today: Recursive Patterns

Today: Recursive Design/Patterns via Examples:

- Working with Lists
- Word Shuffling
- Making Change
- Forming/Evaluating Abstract Syntax Trees