Go watch Inception! (Movie about recursion)

Computational Concepts today

• Variable Scope (also: see reading)
• Recursion

Variable Scope

When an input is passed to a function, what does the function actually get?
- Internal variables get a copy of input values, with the exception of mutable objects

Local variables only exist within the function in which they are defined
- The variables cease to exist when the function ends
- The scope of a variable is the part(s) of code where that variable name binding is valid (i.e. where it exists)

Remember: Functions

```python
def concat(str1, str2):
    return str1+str2
```
**Variable Scope: Example I**

```python
i = 1
def foo():
    i = 5
    print(i, 'in foo()')
print(i, 'global')
foo()
```

Output?

```text
1=global
5 in foo()
```

**Variable Scope: Example II**

```python
a_var = 'global value'
def a_func():
    global a_var
    a_var = 'local value'
    print(a_var, 'in a_func()')
a_func()
print(a_var, 'outside a_func()')
```

Output?

```text
global value
local value
local value
```

**Recursion**

```
A recursive function calls itself, directly or indirectly
```

**Reminder: Iteration**

```
<initialization statements>
for <variables> in <sequence expression>:
    <body statements>
<rest of the program>
```

```
<initialization statements>
while <predicate expression>:
    <body statements>
<rest of the program>
```

```
[ <expr with loop var> for <loop var> in <sequence expr> ]
```
**Iteration vs Recursion**

**Recursion:**
```python
def sum(n):
    if n==0:
        return 0
    return n+sum(n-1)
```

**Why does it work?**

```
sum(3)
# sum(3) -> 3 + sum(2)
#     -> 3 + sum(2) + sum(1)
#     -> 3 + sum(2) + 1 + 0
#     -> 3 + 3 + 1
#     -> 6
```

**How does it work?**

- Each recursive call gets its own local variables
  - Just like any other function call
- Computes its result (possibly using additional calls)
  - Just like any other function call
- Returns its result and returns control to its caller
  - Just like any other function call
- The function that is called happens to be itself
  - Called on a simpler problem
  - Eventually bottoms out on the simple base case
- Reason about correctness “by mathematical induction”
  - Solve a base case
  - Assuming a solution to a smaller problem, extend it

**Local variables**

```python
def sum(n):
    if n==0:
        return 0
    return n+sum(n-1)
```

Each call has its own “frame” of local variables

**Sanity Check...**

- Recursion is more powerful than
  - a) more powerful than
  - b) just as powerful as
  - c) less powerful than
Why Recursion?
• “After Abstraction, Recursion is probably the 2nd biggest idea in this course”
• “It’s tremendously useful when the problem is self-similar”
• “It’s no more powerful than iteration, but often leads to more concise & better code”
• “It’s more ‘mathematical’”
• “It embodies the beauty and joy of computing”
• …

Why Recursion? More Reason
• Recursive structures exist (sometimes hidden) in nature and therefore in data!
• It’s mentally and sometimes computationally more efficient to process recursive structures using recursion.

Recursion (unwanted)

Example I
List all items on your hard disk

List Files in Python
def listfiles(directory):
    content = os.path.join(directory, x) for x in os.listdir(directory)
dirs = sorted([x for x in content if os.path.isdir(x)])
files = sorted([x for x in content if os.path.isfile(x)])
for d in dirs:
    listfiles(d)
for f in files:
    print f

Iterative version about twice as much code and much harder to think about.

Example II
Sort the numbers in a list.

Hidden recursive structure: Decision tree!
Tree Recursion makes Sorting Efficient

Break the problem into multiple smaller subproblems, and solve them recursively

def split(x, s):
    return [i for i in s if i <= x], [i for i in s if i > x]
def qsort(s):
    """Sort a sequence - split it by the first element, sort both parts and put them back together."""
    if not s:
        return []
    else:
        pivot = first(s)
        lessor, more = split(pivot, rest(s))
        return qsort(lessor) + [pivot] + qsort(more)

>>> qsort([3, 3, 1, 4, 5, 4, 3, 2, 1, 17])
[1, 1, 2, 3, 3, 3, 4, 4, 5, 17]

QuickSort Example

Questions?

9/9/16 UCB CS88 Fa16 L3

9/9/16 UCB CS88 Fa16 L3