Programming in Python

BBM103 Introduction to Programming Lab 1
Week 8

Fall 2018
In order to understand recursion, you must understand recursion.
WHAT IS RECURSION?

• Goal: simplify the problem by solving the same problem for smaller input
  • Solve problems by divide(decrease)-and-conquer

• Function calls itself (but not infinitely!)
  • One or more base cases
ITERATION vs. RECURSION

• An **ITERATIVE** function is one that loops to repeat some part of the code.

• A **RECURSIVE** function is one that calls itself again to repeat the code.
Multiplication Example: ITERATIVE Solution

\[ a \times b \text{ is equal to “add } a \text{ to itself } b \text{ times”} \]

\[ a \times b = a + a + a + a + \ldots + a \]

\[ \text{b times} \]

```python
def multiply_iterative(a, b):
    result = 0
    while b > 0:
        result += a
        b -= 1
    return result
```

Iteration
Multiplication Example: RECURSIVE Solution

\[ a \times b = a + a + a + a + \ldots + a = a + a \times (b-1) \]

\[ \text{b times} \]

\[ \text{b-1 times} \]

def mult_recursive(a, b):
    if b == 1:
        return a
    else:
        return a + mult_recursive(a, b-1)

Base case

Recursive Step
Factorial Example: ITERATIVE Solution

\[ n! = n \times (n-1) \times (n-2) \times (n-3) \times \ldots \times 1 \]

```python
def factorial_iterative(n):
    result = 1
    while n > 0:
        result *= n
        n -= 1
    return result
```

Iteration
Factorial Example: RECURSIVE Solution

\[
n! = n \times (n-1) \times (n-2) \times (n-3) \times \ldots \times 1
\]

- **Base Case:** \( \text{if } n = 1 \rightarrow 1! = 1 \)
- **Recursive step:** \( n! = n \times (n-1)! \)

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

Base case

Recursive Step
ITERATION vs. RECURSION

• recursion may be simpler, more intuitive, and also efficient and natural for a programmer.

• BUT! Recursion may not be efficient from the computer’s point of view.
  
  • Ex. Computing $n^{th}$ Fibonacci number recursively takes $O(2^n)$ steps!
Example: Fibonacci Numbers

The Fibonacci numbers are the numbers of the following sequence of integer values:
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...
The Fibonacci numbers are defined by:

\[ F_n = F_{n-1} + F_{n-2} \]

with \( F_0 = 0 \) and \( F_1 = 1 \)

```python
def fibonacci(n):
    a, b = 0, 1
    for i in range(n):
        a, b = b, a + b
    return a

number = input("Please enter a number to print fibonacci numbers!")
print(fibonacci(int(number)))
```

Output:
Please enter a number to print fibonacci numbers!4
3
Example: Visualizing Recursion

```python
34 import turtle
35 
36 def tree(branchLen,t):
37     if branchLen > 5:
38         t.forward(branchLen)
39         t.right(20)
40         tree(branchLen-15,t)
41         t.left(40)
42         tree(branchLen-15,t)
43         t.right(20)
44         t.backward(branchLen)
45 
46 def main():
47     t = turtle.Turtle()
48     myWin = turtle.Screen()
49     t.left(90)
50     t.up()
51     t.backward(100)
52     t.down()
53     t.color("green")
54     tree(75,t)
55     myWin.exitonclick()
56 
57 main()
```

Output:
Example: Computing Exponent

```python
def exp(x, n):
    ""
    Computes the result of x raised to the power of n.
    ""
    if n == 0:
        return 1
    else:
        return x * exp(x, n-1)
```

We can compute exponent in fewer steps if we use successive squaring.

```python
number1=input("print a number as base")
number2=input("print a number as exponent")
print(exp(int(number1),int(number2)))
```

```python
def fast_exp(x, n):
    if n == 0:
        return 1
    elif n % 2 == 0:
        return fast_exp(x*x, n/2)
    else:
        return x * fast_exp(x, n-1)
```

Let's look at the execution pattern now.

```python
fast_exp(2, 10)
```

```plaintext
fast_exp(4, 5) # 2 * 2
```

```plaintext
fast_exp(16, 2) # 4 * 4
```

```plaintext
fast_exp(256, 1) # 16 * 16
```

```plaintext
fast_exp(256, 0)
```

```plaintext
1
```

```plaintext
256
```

```plaintext
256 * 1
```

```plaintext
256
```

```plaintext
16
```

```plaintext
4
```

```plaintext
2
```

```plaintext
8
```

```plaintext
16
```

```plaintext
256
```

```plaintext
1024
```

```plaintext
1024
```

```plaintext
1024
```
Example: Flatten a List

```python
def flatten_list(a, result=None):
    if result is None:
        result = []
    for x in a:
        if isinstance(x, list):
            flatten_list(x, result)
        else:
            result.append(x)
    return result

listToFlat=[[1, 2, [3, 4]], [5, 6], 7]
print(listToFlat)
faltList=flatten_list(listToFlat)
print(faltList)
```

Output:

```python
[[[1, 2, [3, 4]], [5, 6], 7]
[1, 2, 3, 4, 5, 6, 7]]
```