CHAPTER 15
RECURSION

ACKNOWLEDGEMENT: THESE SLIDES ARE ADAPTED FROM SLIDES PROVIDED WITH INTRODUCTION TO PROGRAMMING USING PYTHON, LIANG (PEARSON 2013)
OVERVIEW

• **Recursion** is an algorithmic technique where a function calls itself directly or indirectly.

• Why learn recursion?
  • New mode of thinking.
  • Powerful programming paradigm.

• Many computations are naturally self-referential.
  • A folder containing files and other folders.
  • Mathematical sequences - \( f_{n+1} = f_n + 1 \) (whole numbers)
  • Exploring mazes – make a step in the maze, then keep exploring the maze
EXAMPLE
COUNTING

• Lets take an example of counting from 0
• The next number is the previous number plus 1, or mathematically:
  \[ f_n = f_{n-1} + 1, \text{ where } f_0 = 0 \]
• So lets compute \( f_5 \)
  • \( f_5 = f_4 + 1 \), this would be great if we knew \( f_4 \), so lets expand it
  • \( f_5 = (f_3 + 1) + 1 \), then
  • \( f_5 = ((f_2 + 1) + 1) + 1 \), then
  • \( f_5 = (((f_1 + 1) + 1) + 1) + 1 \), then
  • \( f_5 = (((f_0 + 1) + 1) + 1) + 1 \), then finally
  • \( f_5 = (((0 + 1) + 1) + 1) + 1 \) + 1 = 5
PRACTICE RECURSIVE FORMULAS
FIBONACCI SEQUENCE

• The Fibonacci Sequence is used in various places in mathematics and computer science
  \[ f_n = f_{n-1} + f_{n-2}, \text{ where } f_0 = 0, f_1 = 1 \]
• Expand and evaluate \( f_6 \), work with a partner and show your work
HOW DO WE DO RECURSION IN CODE?

• Simply call the function within its own body

```python
def foo():
    # possibly do some stuff
    foo()  # This example of recursion
    # possibly do some more stuff
```
# This function counts using recursion

def recursiveCount(n):
    # \( f_0 = 0 \)
    if n == 0:
        return 0
    # \( f_n = f_{n-1} + 1 \)
    return recursiveCount(n+1) + 1

Together lets trace

\[
\text{print}(\text{recursiveCount}(3))
\]

\[
\text{recursiveCount}(3)
\]

\[
\text{return} \ \text{recursiveCount}(2) + 1
\]

\[
\text{return} \ \text{recursiveCount}(1) + 1
\]

\[
\text{return} \ \text{recursiveCount}(0) + 1
\]

\[
\text{return} \ 0
\]

\[
\text{return} \ 0 + 1
\]

\[
\text{return} \ 1 + 1
\]

\[
\text{return} \ 2 + 1
\]

\[
3
\]
PRACTICE RECURSIVE CODE
FIBONACCI SEQUENCE

• Write a Python function Fibonacci for
  \[ f_n = f_{n-1} + f_{n-2} \]

  ```python
  def Fibonacci(n):
    if n == 0:
      return 0
    if n == 1:
      return 1
    return Fibonacci(n-1) + Fibonacci(n-2)
  ```

• Practice tracing Fibonacci(3)
  ```
  Fibonacci(3)
  • return Fibonacci(2) + Fibonacci(1)
  • return Fibonacci(1) + Fibonacci(0)
  • return 1
  • return 1 + Fibonacci(0)
  • return 0
  • return 1 + 0
  • return 1 + Fibonacci(1)
  • return 1
  • return 1 + 1
  • 2
  ```
CHARACTERISTICS OF RECURSION

• All recursive methods have the following characteristics:
  • One or more base cases (the simplest case) are used to stop recursion.
  • Every recursive call reduces the original problem, bringing it increasingly closer to a base case until it becomes that case.

• In general, to solve a problem using recursion, you break it into subproblems. If a subproblem resembles the original problem, you can apply the same approach to solve the subproblem recursively. This subproblem is almost the same as the original problem in nature with a smaller size.
DESIGNING RECURSIVE FUNCTIONS

• Identify the base case
  • The base case is the part of the recursion not defined in terms of itself, e.g., $f_0 = 0, f_1 = 1$
  • This is when the recursion stops! If you forget your base case, then the world will end
    • Really an infinite series of function calls until your computer crashes (if it ever does)

• Identify the recursive process
  • This is the algorithmic process or algorithmic formula

• Write the code
PRACTICE DESIGNING RECURSIVE FUNCTIONS
GREATEST COMMON DENOMINATOR (GCD)

• GCD
  • For two integers \( p \) and \( q \), if \( q \) divides \( p \) then the GCD of \( p \) and \( q \) is \( q \)
  • Otherwise the GCD of \( p \) and \( q \) is the same as \( q \) and \( p \% q \)

• Step 1: Identify the base case
  • \( q = 0 \) implies that the GCD is \( p \)

• Step 2: Identify the recursive step
  • \( GCD(q, p \% q) \)

• Step 3: Code

1. \texttt{def gcd(p, q):}
2. \hspace{1em} 	exttt{if q == 0:}
3. \hspace{2em} 	exttt{return p}
4. \hspace{1em} 	exttt{return gcd(q, p \% q)
RECURSIVE GCD DEMO

1. def gcd(p, q):
2.   if q == 0:
3.     return p
4.   return gcd(q, p % q)
5.
6. print(gcd(1272, 216));
1. `def gcd(p, q):
2.     if q == 0:
3.         return p
4.     return gcd(q, p % q)

`gcd(1272, 216)`

Memory gcd call - 1

$p = 1272, q = 216$
```python
def gcd(p, q):
    if q == 0:
        return p
    return gcd(q, p % q)
```

Memory gcd call - 1

$p = 1272, q = 216$

gcd(1272, 216)

1. def gcd(p, q):
2.     if q == 0:
3.         return p
4.     return gcd(q, p % q)
```python
1. def gcd(p, q):
2.     if q == 0:
3.         return p
4.     return gcd(q, p%q)

p = 1272, q = 216
Memory gcd call - 1

gcd(1272, 216)
```
```python
1. def gcd(p, q):
2.     if q == 0:
3.         return p
4.     return gcd(q, p % q)
```

1. gcd(1272, 216)

```
p = 1272, q = 216
```

Memory gcd call - 1

```
1272 = 216 \times 5 + 192
```

```
p = 1272, q = 216
```

```
p = 216, q = 192
```

Memory gcd call - 2

```
gcd(216, 192)
```

```
p = 216, q = 192
```

```
gcd(216, 192)
```

```
gcd(216, 192)
```

```text
1272 = 216 \times 5 + 192
```
1. def gcd(p, q):
2.     if q == 0:
3.         return p
4.     return gcd(q, p%q)

p = 1272, q = 216

Memory gcd call - 1

p = 216, q = 192

Memory gcd call - 2

gcd(216, 192)

1. def gcd(p, q):
2.     if q == 0:
3.         return p
4.     return gcd(q, p%q)
1. def gcd(p, q):
2.     if q == 0:
3.         return p
4.     return gcd(q, p%q)

gcd(1272, 216)

p = 1272, q = 216
Memory gcd call - 1

1. def gcd(p, q):
2.     if q == 0:
3.         return p
4.     return gcd(q, p%q)

gcd(216, 192)

p = 216, q = 192
Memory gcd call - 2
1. `def gcd(p, q):
2.     if q == 0:
3.         return p
4.     return gcd(q, p%q)`

- gcd(1272, 216)
  - p = 1272, q = 216
  - Memory gcd call - 1

- gcd(216, 192)
  - p = 216, q = 192
  - Memory gcd call - 2

- gcd(192, 24)
  - p = 192, q = 24
  - Memory gcd call - 3
1. def gcd(p, q):
   2.     if q == 0:
   3.         return p
   4.     return gcd(q, p % q)

gcd(1272, 216)

1. def gcd(p, q):
   2.     if q == 0:
   3.         return p
   4.     return gcd(q, p % q)

gcd(216, 192)

1. def gcd(p, q):
   2.     if q == 0:
   3.         return p
   4.     return gcd(q, p % q)

gcd(192, 24)
def gcd(p, q):
    if q == 0:
        return p
    return gcd(q, p % q)

p = 1272, q = 216

Memory gcd call - 1

gcd(1272, 216)

gcd(216, 192)

Memory gcd call - 2

p = 216, q = 192

gcd(192, 24)

Memory gcd call - 3

p = 192, q = 24
```python
1. def gcd(p, q):
2.     if q == 0:
3.         return p
4.     return gcd(q, p%q)
```

1. gcd(1272, 216)

Memory gcd call - 1

- p = 1272, q = 216

1. gcd(216, 192)

Memory gcd call - 2

- p = 216, q = 192

1. gcd(192, 24)

Memory gcd call - 3

- p = 192, q = 24

1. gcd(24, 0)

Memory gcd call - 4

- p = 24, q = 0
1. def gcd(p, q):
2.     if q == 0:
3.         return p
4.     return gcd(q, p%q)

gcd(1272, 216)

p = 1272, q = 216

Memory gcd call - 1

gcd(216, 192)

p = 216, q = 192

Memory gcd call - 2

gcd(192, 24)

p = 192, q = 24

Memory gcd call - 3

gcd(24, 0)

p = 24, q = 0

Memory gcd call - 4
```python
1. def gcd(p, q):
   2.     if q == 0:
   3.         return p
   4.     return gcd(q, p%q)
```

1. gcd(1272, 216)
   Memory gcd call - 1

1. p = 1272, q = 216

2. gcd(216, 192)
   Memory gcd call - 2

1. p = 216, q = 192

3. gcd(192, 24)
   Memory gcd call - 3

1. p = 192, q = 24

4. gcd(24, 0)
   Memory gcd call - 4

1. p = 24, q = 0

24
1. def gcd(p, q):
2.     if q == 0:
3.         return p
4.     return gcd(q, p%q)

gcd(1272, 216)
p = 1272, q = 216

Memory gcd call - 1

p = 216, q = 192
Memory gcd call - 2

gcd(216, 192)
p = 216, q = 192

Memory gcd call - 2

p = 192, q = 24
Memory gcd call - 3

gcd(192, 24)
p = 192, q = 24

Memory gcd call - 3

24
```python
def gcd(p, q):
    if q == 0:
        return p
    return gcd(q, p % q)
```

Call Stack:
1. `gcd(1272, 216)`
   - `p = 1272, q = 216`
   - `memory gcd call - 1`

2. `gcd(216, 192)`
   - `p = 216, q = 192`
   - `memory gcd call - 2`

3. `gcd(192, 24)`
   - `p = 192, q = 24`
   - `memory gcd call - 3`

4. `gcd(24, 24)`
   - `return 24`
1. def gcd(p, q):
2.     if q == 0:
3.         return p
4.     return gcd(q, p%q)

gcd(1272, 216)

1. def gcd(p, q):
2.     if q == 0:
3.         return p
4.     return gcd(q, p%q)

gcd(1272, 216)

1. def gcd(p, q):
2.     if q == 0:
3.         return p
4.     return gcd(q, p%q)

gcd(216, 192)

p = 216, q = 192

Memory gcd call - 2

p = 1272, q = 216

Memory gcd call - 1
1. def gcd(p, q):
2.     if q == 0:
3.         return p
4.     return gcd(q, p%q)

gcd(1272, 216)

- Memory gcd call - 1

p = 1272, q = 216

24

gcd(216, 192)

- Memory gcd call - 2

p = 216, q = 192

24

24
def gcd(p, q):
    if q == 0:
        return p
    return gcd(q, p % q)

gcd(1272, 216)

Memory gcd call - 1

p = 1272, q = 216
1. def gcd(p, q):
2.     if q == 0:
3.         return p
4.     return gcd(q, p%q)
5. 
6. print(gcd(1272, 216));
Many of the problems presented in the early chapters can be solved using recursion if you think recursively. For example, the palindrome problem can be solved recursively as follows:

```python
1. def isPalindrome(s):
2.     if s.length() <= 1:        # Base case
3.         return True
4.     elif s[0] != s[len(s) - 1]:  # Base case
5.         return False
6.     else:
7.         return isPalindrome(s[1:len(s)-1])
```
RECURSIVE HELPER METHODS

The preceding recursive `isPalindrome` method is not efficient, because it creates a new string for every recursive call. To avoid creating new strings, use a helper method:

1. `def isPalindrome(s):
2.    return isPalindromeHelper(s, 0, len(s) - 1)
3.
4. `def isPalindromeHelper(s, low, high):
5.    if high <= low:  # Base case
6.        return True
7.    elif s[low] != s[high]:  # Base case
8.        return False
9.    else:
10.       return isPalindromeHelper(s, low + 1, high - 1)
DOWNSIDE OF RECURSION

• Recursion is not always efficient!
• Take for instance, the Fibonacci sequence

```python
def F(n):
    if n == 0:
        return 0
    if n == 1:
        return 1
    return F(n-1) + F(n-2)
```

• F(50) is called once.
• F(49) is called once.
• F(48) is called 2 times.
• F(47) is called 3 times.
...
• F(1) is called 12,586,269,025 times.
BEST PRACTICE
CONVERT RECURSIVE ALGORITHMS TO ITERATIVE ONES

• Try to do this whenever possible
• Example Fibonacci Sequence

1. # This is an example conversion. You can be even more efficient!
2. def F(n):
3.   if n <= 1:
4.     return n
5.   fn2, fn1, fn = 0, 1, 1
6.   # Iterative means repetition until failure condition,
7.   # typically done with loops and not recursion
8.   for i in range(2, n+1):
9.     fn = fn1 + fn2;
10.    fn2, fn1 = fn1, fn
11.   return fn
OR DO TAIL RECURSION

• Tail recursion is when the last operation of a function is the recursive call

• Example with factorial:
  \[ f_n = n \times f_{n-1} \]

1. # Without tail recursion
2. def factorial(n):
3.   if n == 0:
4.     return 1
5.   else:
6.     return n*factorial(n-1)

1. # With tail recursion
2. def factorial(n):
3.   return factorial(n, 1)
4. def factorial(n, result):
5.   if n == 0:
6.     return result
7.   else:
8.     return factorial(n-1, n * result)
SUMMARY

• How to write simple recursive programs?
  • Base case, reduction step.
• Trace the execution of a recursive program.
• Why learn recursion?
  • New mode of thinking.
  • Powerful programming tool
TOWERS OF HANOI

PRACTICE
TOWERS OF HANOI

• Design recursive algorithm to move all the discs from the leftmost peg to the rightmost one.
  • Only one disc may be moved at a time.
  • A disc can be placed either on empty peg or on top of a larger disc.
SOLUTION
TOWERS OF HANOI

Move n-1 smallest discs right.

Move largest disc right.

Move n-1 smallest discs right.
TOWERS OF HANOI

1. def moves(n, from, to, aux):
2.     if n == 0:
3.         return
4.     moves(n-1, from, aux, to)
5.     print("Move disk", n, "from peg", from, "to peg", to)
6.     moves(n-1, aux, to, from)
7.
8. moves(3, 'A', 'C', 'B')
RECURSION TREE (FUNCTION TRACE)
TOWERS OF HANOI