DEPAUL UNIVERSITY
COLLEGE OF COMPUTING AND DIGITAL MEDIA

CSC 243 NOTES
Fall 2020: Amber Settle

Monday, October 19, 2020

Announcements

- Midterm
  - I will post a recording giving the solution to the midterm ASAP
  - I will have the midterm submissions graded by the end of the week
- The fifth assignment is due Thursday, October 22nd – questions?

Recursion

A **recursive method** is a method that solves a problem by making one or more calls to itself.

Recursion is a particularly helpful tool for certain problems.

Some programming languages are even primarily recursive in nature (e.g. Lisp and Scheme), where loops are almost completely absent from programs written in them.

Any recursive method consists of:

- **One or more base cases**: these are the portions of the problem for which an immediate solution is known
- **One or more recursive calls**: to avoid infinite recursion these subproblems must be in some way smaller than the original problem

The best way to learn recursion is to practice, a LOT. So we will see many examples during this part of the course.

Printing numbers

Consider the problem of printing the numbers from n to 1 for a positive value of n.

Input: A non-negative integer n
Output: The numbers from n down to 1, printed one per line

Sample runs:

```
>>> countdown(10)
10
9
8
```

```
7
6
5
4
3
2
1
Blast off!
>>> countdown(1)
1
Blast off!
>>> countdown(0)
Blast off!
>>> countdown(-3)
Blast off!
>>>
```

We will write a **recursive algorithm** to solve the problem.

To do that we need to think recursively, meaning that we need to construct subproblems that when solved and combined can produce the solution to the original problem. We also need to find what the base cases are.

Let's first think about the **base case**. When would the problem be easy?

If we had no numbers to print, we could just print the "Blast off!" message.

How can we use that to find a **recursive solution**?

- We could print the first number we are required to print.
- Then we could make a recursive call to print any other numbers as well as the "Blast off!"message.

See the solution in **numbers.py**.

Printing vertically

Consider the problem of printing an integer's digits vertically:

Input: A non-negative integer n
Output: The integer n, with its digits stacked vertically

Example: If the input is 1234, the output is:

1
2
3
4

We will write a **recursive algorithm** to solve the problem.

To do that we need to think recursively, meaning that we need to construct subproblems that when solved and combined can produce the solution to the original problem. We also need to find what the base cases are.

Let's first think about the **base case**. When would the problem be easy?

If we had just one digit to print, then we would print it on a line by itself and would be done.

How can we use that to find a **recursive solution**?

- We could print all but the last digit by making a recursive call. (Why the last digit and not the first one?)
- Then we could print the last digit. (Why not print the last digit before the recursive call?)

See the file **vertical.py** for the solution.

Why and how does this solution work?

What happens when we call vertical on input n = 361?

Executing **vertical(361)**:

Since 361 is not less than 10, the call to vertical(36) is made.

Before this call gets executed the computer must store all the information necessary to complete the original call vertical(361) after vertical(36) is completed.

The necessary information includes the values of the variables and where execution should resume. It is stored in an **activation record**.

So before executing vertical(36), we store the activation record:

n = 361
return at step 6

Executing **vertical(36)**:

Since 36 is not less than 10, the call to vertical(3) is made.

Before doing this we store the activation record:

n = 36
return at step 6

Executing **vertical(3)**:

Since 3 is less than 10, we output 3 and return … where?

Back to line 6 in the execution of vertical(36).

Executing **vertical(36)**:

We output 6 and return to line 6 of the execution of vertical(361).

Executing **vertical(361)**:

We output 1 and stop.

Exercises

Problem 1: Consider a reversed version of vertical print.

Input: A non-negative integer n
Output: The integer n, with its digits stacked vertically, in reverse order.

Example: If n = 361, then the program should output:

1
6
3

Hint: How can we modify the recursive solution for the previous problem to get the solution to this problem?

Problem 2: Write a recursive function **printLst**() that takes a list as a parameter and prints the items in the list, one per line.

```
>>> printLst([1, 2, 3])
1
2
3
>>> printLst([1])
1
>>> printLst([])
```

```
>>>
```

Problem 3: Write a recursive function **cheer**() that on an integer parameter n, will output n-1 strings “Hip” followed by “Hurrah”.

```
>>> cheer(5)
Hip
Hip
Hip
Hip
Hurrah
>>> cheer(3)
Hip
Hip
Hurrah
>>> cheer(1)
Hurrah
>>> cheer(2)
Hip
Hurrah
>>> cheer(0)
Hurrah
```