Recursion

COSC 101, 2018-04-09

Announcements
● Exam 2 Thursday @ 7pm in 207 Lathrop Hall
● No office hours Wednesday

Outline
● Computational thinking
● Recursion
● Example: downup

Computational thinking
● Write an algorithm to count how many people there are between you and the left wall of the classroom, including yourself.
  ```python
  count = 0
  for desk in your_row:
    if person in desk:
      if person == you:
        return count + 1
      else:
        count = count + 1
  ```
● Now assume the person sitting to your left could help you count how many people there are between you and the left wall of the classroom. What would you ask that person to do? (Assume there is a person to your left.)
  ○ Ask them to count the number of people to their left, including themself, and tell you the count
  ○ (Wait while they count)
  ○ You add one to their count so the count also includes yourself
● What should you do if there is no person to your left? --- say the number of people is 1 (i.e., just yourself)
  ```python
  Pseudocode:
  if not person_to_left:
    return 1
  else:
    count = ask_person_to_left()
    return count + 1
  ```
● What if we wanted to count the total number of people in the row?
  ○ The rightmost person in the row executes our algorithm first
  ○ Everyone in the row is following the same algorithm, so we’ll put it in a function so everyone in the row can “re-use” the same instructions
  ```python
  def ask_person_to_left():
    if not person_to_left:
      return 1
    else:
      count = ask_person_to_left()
      return count + 1
  ```
● Congratulations, you just wrote your first recursive function!
Recursion

- A method of solving a problem that breaks the problem into smaller and smaller subproblems until the subproblem is trivial to solve
  - E.g., add one to left neighbor’s count; if person has no left neighbor, they answer 1
- Usually involves a function calling itself --- e.g., ask_person_to_left()
  - You can also have mutual recursion in which functions call each other --- we won’t discuss this

Example: downup

- Write a function called downup that takes a string as a parameter and prints the entire string, then all but the last letter, than all but the second to last letter, … then only the first letter, then the first and second letter, … then the entire string. For example, downup("monday") would print:
  
  - monad
  - mond
  - mon
  - mo
  - m
  - mo
  - mond
  - monda
  - Monday
  
  As another example, downup("x") would print:
  
  - x

Your function should not be recursive. Hint: you’ll need at least two for loops.

```python
def downup(string):
    for i in range(len(string), 1, -1):
        print(string[:i])
    for i in range(1, len(string)+1):
        print(string[:i])
```

- This is an iterative solution --- i.e., it uses loops; it is not recursive
- We want to write a recursive solution
  - Writing recursive functions is tricky, but you’ll get good at with patience and practice
  - Start with a non-iterative, non-recursive function that works for some strings

- Write a function called downup1 that takes a string with a single character as a parameter and produces the same output as downup would for a single-character string.
  
  ```python
def downup1(string):
    print(string)
```

- Now, write a function called downup2 that takes a string with two characters as a parameter and produces the same output as downup would for a two-character string.
  
  ```python
def downup2(string):
    print(string)
    print(string[:1])
    print(string)
```

- What output does your downup2 function produce if the provided string is more than two characters long: e.g., "four"?
Rewrite your downup2 function so it ignores any characters beyond the first two.

```python
def downup2(string):
    print(string[:2])
    print(string[:1])
    print(string[:2])
```

Now, write a function called downup3 that takes a string with three characters as a parameter and produces the same output as downup would for a three-character string. If the string is longer than three characters, your function should ignore the extra characters.

```python
def downup3(string):
    print(string[:3])
    print(string[:2])
    print(string[:1])
    print(string[:2])
    print(string[:3])
```

Are your downup2 and downup3 functions DRY (i.e., Don’t Repeat Yourself)? If necessary, revise your downup3 function so it is DRY.

```python
def downup3(string):
    print(string[:3])
downup2(string)
    print(string[:3])
```

Write a downup4 function that is DRY.

```python
def downup4(string):
    print(string[:4])
downup3(string)
    print(string[:4])
```

What do you notice about your downup3 and downup4 functions?
Same, except for the end index of the slice

Write a function called downupN that is a more general version of downup3 and downup4.

```python
def downupN(string, n):
    print(string[:n])
downupN(string,n-1)
    print(string[:n])
```

Is your function a suitable replacement for downup2? How about downup1?
Can replace downup2 with downupN, but cannot replace downup1

Try to modify downupN so it also works when n is 1.

```python
def downupN(string, n):
    if n == 1:
        print(string)
    else:
        print(string[:n])
downupN(string,n-1)
        print(string[:n])
```

Now try to modify your function you eliminate the parameter n and use the len function.

```python
def downupN(string):
    if len(string) == 1:
```
print(string)
else:
    print(string[:len(string)])
    downupN(string[:len(string)-1])
    print(string[:len(string)])

• Recursive calls for "monday"
    downupN("monday")
    print("monday")
    downupN("monda")
    print("monda")
    downupN("mond")
    print("mond")
    downupN("mon")
    print("mon")
    downupN("mo")
    print("mo")
    downupN("m")
    print("m")
    print("mo")
    print("mon")
    print("mond")
    print("monda")
    print("monday")

• Congratulations, you just wrote your second recursive function!