Lab 4 Solutions lab04.zip (lab04.zip)

Solution Files

Topics

Consult this section if you need a refresher on the material for this lab. It's okay to skip directly to the questions and refer back here should you get stuck.

Required Questions

Recursion

Q1: Skip Add

Write a function `skip_add` that takes a single argument `n` and computes the sum of every other integer between `0` and `n`. Assume `n` is non-negative.

```
this_file = __file__

def skip_add(n):
    """ Takes a number n and returns n + n-2 + n-4 + n-6 + ... + 0.

    >>> skip_add(5)  # 5 + 3 + 1 + 0
    9
    >>> skip_add(10) # 10 + 8 + 6 + 4 + 2 + 0
    30
    >>> # Do not use while/for loops!
    >>> from construct_check import check
    >>> # ban iteration
    >>> check(this_file, 'skip_add',
    ...       ['While', 'For'])
    True
    """
    if n <= 0:
        return 0
    return n + skip_add(n - 2)
```

Use Ok to test your code:

```
python3 ok -q skip_add
```

Q2: Summation

Now, write a recursive implementation of `summation`, which takes a positive integer `n` and a function `term`. It applies `term` to every number from `1` to `n` including `n` and returns the sum of the results.

```
def summation(n, term):

    """Return the sum of the first n terms in the sequence defined by term.
    Implement using recursion!

    >>> summation(5, lambda x: x * x * x) # 1^3 + 2^3 + 3^3 + 4^3 + 5^3
    225
    >>> summation(9, lambda x: x + 1) # 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10
    54
    >>> summation(5, lambda x: 2**x) # 2^1 + 2^2 + 2^3 + 2^4 + 2^5
    62
    >>> # Do not use while/for loops!
    >>> from construct_check import check
    >>> # ban iteration
    >>> check(this_file, 'summation',
    ...       ['While', 'For'])
    True
    """
    assert n >= 1
    if n == 1:
        return term(n)
    else:
        return term(n) + summation(n - 1, term)
    # Base case: only one item to sum, so we return that item.
    # Recursive call: returns the result of summing the numbers up to n-1 using
    # term. All that's missing is term applied to the current value n.
```

Use Ok to test your code:

```
python3 ok -q summation
```

Q3: GCD

The greatest common divisor of two positive integers `a` and `b` is the largest integer which evenly divides both numbers (with no remainder). Euclid, a Greek mathematician in 300 B.C., realized that the greatest common divisor of `a` and `b` is one of the following:

- the smaller value if it evenly divides the larger value, or
- the greatest common divisor of the smaller value and the remainder of the larger value divided by the smaller value

In other words, if `a` is greater than `b` and `a` is not divisible by `b`, then

```
gcd(a, b) = gcd(b, a % b)
```

Write the `gcd` function recursively using Euclid's algorithm.

```
def gcd(a, b):
    """Returns the greatest common divisor of a and b.
    Should be implemented using recursion.

    >>> gcd(34, 19)
    1
    >>> gcd(39, 91)
    13
    >>> gcd(20, 30)
    10
    >>> gcd(40, 40)
    40
    """
    a, b = max(a, b), min(a, b)
    if a % b == 0:
        return b
    else:
        return gcd(b, a % b)

# Iterative solution, if you're curious
def gcd_iter(a, b):
    """Returns the greatest common divisor of a and b, using iteration.

    >>> gcd_iter(34, 19)
    1
    >>> gcd_iter(39, 91)
    13
    >>> gcd_iter(20, 30)
    10
    >>> gcd_iter(40, 40)
    40
    """
    if a < b:
        return gcd_iter(b, a)
    while a > b and not a % b == 0:
        a, b = b, a % b
    return b

    # Video Walkthrough: https://youtu.be/yBhhfBObNxs
```

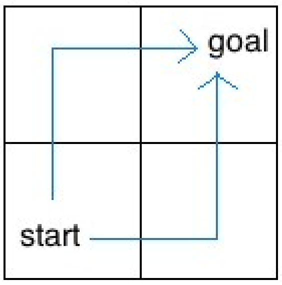
Use Ok to test your code:

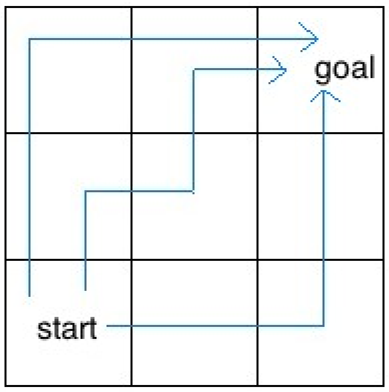
```
python3 ok -q gcd
```

Tree Recursion

Q4: Insect Combinatorics

Consider an insect in an *M* by *N* grid. The insect starts at the bottom left corner, *(0, 0)*, and wants to end up at the top right corner, *(M-1, N-1)*. The insect is only capable of moving right or up. Write a function `paths` that takes a grid length and width and returns the number of different paths the insect can take from the start to the goal. (There is a closed-form solution (https://en.wikipedia.org/wiki/Closed-form_expression) to this problem, but try to answer it procedurally using recursion.)





For example, the 2 by 2 grid has a total of two ways for the insect to move from the start to the goal. For the 3 by 3 grid, the insect has 6 diferent paths (only 3 are shown above).

```
def paths(m, n):
    """Return the number of paths from one corner of an
    M by N grid to the opposite corner.

    >>> paths(2, 2)
    2
    >>> paths(5, 7)
    210
    >>> paths(117, 1)
    1
    >>> paths(1, 157)
    1
    """
    if m == 1 or n == 1:
        return 1
    return paths(m - 1, n) + paths(m, n - 1)
```

Use Ok to test your code:

```
python3 ok -q paths
```

Q5: Maximum Subsequence

A subsequence of a number is a series of (not necessarily contiguous) digits of the number. For example, 12345 has subsequences that include 123, 234, 124, 245, etc. Your task is to get the maximum subsequence below a certain length.

```
def max_subseq(n, l):
    """
    Return the maximum subsequence of length at most l that can be found in the
    For example, for n = 20125 and l = 3, we have that the subsequences are
        2
        0
        1
        2
        5
        20
        21
        22
        25
        01
        02
        05
        12
        15
        25
        201
        202
        205
        212
        215
        225
        012
        015
        025
        125
    and of these, the maxumum number is 225, so our answer is 225.

    >>> max_subseq(20125, 3)
    225
    >>> max_subseq(20125, 5)
    20125
    >>> max_subseq(20125, 6) # note that 20125 == 020125
    20125
    >>> max_subseq(12345, 3)
    345
    >>> max_subseq(12345, 0) # 0 is of length 0
    0
    >>> max_subseq(12345, 1)
    5
    """
    if n == 0 or l == 0:
        return 0
    with_last = max_subseq(n // 10, l - 1) * 10 + n % 10
    without_last = max_subseq(n // 10, l)
    return max(with_last, without_last)
```

Use Ok to test your code:

```
python3 ok -q max_subseq
```

There are two key insights for this problem

- You need to split into the cases where the ones digit is used and the one where it is not. In the case where it is, we want to reduce `l` since we used one of the digits, and in the case where it isn't we do not.
- In the case where we are using the ones digit, you need to put the digit back onto the end, and the way to attach a digit `d` to the end of a number `n` is `10 * n + d`.

Submit

Make sure to submit this assignment by running:

```
python3 ok --submit
```

CS 61A (/)

Weekly Schedule (/weekly.html)

Office Hours (/office-hours.html)

Staff (/staff.html)

Resources (/resources.html)

Studying Guide (/articles/studying.html)

Debugging Guide (/articles/debugging.html)

Composition Guide (/articles/composition.html)

Policies (/articles/about.html)

Assignments (/articles/about.html#assignments)

Exams (/articles/about.html#exams)

Grading (/articles/about.html#grading)