

Python Recursion Workshop 2

CSCI 204

solution

Solve the following problems using recursion. You can work in pair or alone. Use whichever Python environment you feel comfortable.

1. Determine if a non-negative integer **b** is a prime. The basic idea is to check consecutively if **b** is divisible by **b - 1**, **b - 2**, **b - 3**, until 1. This can be done by checking if **b % x == 0**. If any of the **b - i** can divide **b** evenly, then **b** is not a prime, we can stop. If we are able to reach the check **b % 1**, it means **b** is a prime.
For example, for 5, we check $5\%4$, $5\%3$, $5\%2$, until $5\%1$, none is equal to zero, so 5 is a prime.
For example, for 6, we check $6\%5$, $6\%4$, $6\%3$ is equal to zero, so 6 is not a prime.

```
def is_prime(b, x):
    """Check to see if 'b' is a prime. x == b - 1"""
    if x == 1:
        return True
    elif b % x == 0:
        return False
    else:
        return is_prime(b, x - 1)

print('is_prime(5, 4) ', is_prime(5,4))
print('is_prime(13, 12) ', is_prime(13, 12))
print('is_prime(20, 19) ', is_prime(20,19))
print('is_prime(33, 32) ', is_prime(33, 32))
```

2. List all permutations of a string **s**. For example, if we have a string 'abc', the complete list of its permutations are 'abc', 'acb', 'bac', 'bca', 'cab', 'cba'. The idea is to take out one element of the list at a time, make it a part of the prefix which starts as an empty string. Then recursively pursue the step until all elements in the string become a part of the prefix.

E.g., 'abcd'

```
a. 'a' + recursively('bcd')
b. 'b' + recursively('acd')
c. 'c' + recursively('abd')
d. 'd' + recursively('abc')

def list_permutations( prefix, s ):
    """
    List all permutations of a string
    """
    if len(s) == 0:
        print( prefix )
    else:
```

```
    for i in range(len(s)):
        ch = s[i]
        rest = s[0:i] + s[(i + 1):]
        list_permutations( prefix + ch, rest)

list_permutations('', 'XYZ')
#list_permutations('', 'ABCD')
```