Every Recursive function has three things.

1. One or more base cases
2. One or more ways to break the problem down into a smaller problem
   - E.g. Given a number as input, we need to break it down into a smaller number
3. Solve the smaller problem recursively; from that, form a solution to the original problem
1. Complete the definition for `num_digits`, which takes in a number `n` and returns the number of digits it has.

```python
def num_digits(n):
    """Takes in a positive integer and returns the number of digits."""
    if n < 10:
        return 1
    else:
        return 1 + num_digits(n // 10)
```

Solution:
2. Write a function `is_sorted` that takes in an integer `n` and returns true if the digits of that number are increasing from right to left.

```python
def is_sorted(n):
    
    >>> is_sorted(2)
    True
    >>> is_sorted(22222)
    True
    >>> is_sorted(9876543210)
    True
    >>> is_sorted(9087654321)
    False
    
    Solution:
    
    right_digit = n % 10
    rest = n // 10
    if rest == 0:
        return True
    elif right_digit > rest % 10:
        return False
    else:
        return is_sorted(rest)
```
3. Mario needs to jump over a series of Piranha plants, represented as a string of 0’s and 1’s. Mario only moves forward and can either step (move forward one space) or jump (move forward two spaces) from each position. How many different ways can Mario traverse a level without stepping or jumping into a Piranha plant? Assume that every level begins with a 1 (where Mario starts) and ends with a 1 (where Mario must end up).

```python
def mario_number(level):
    """
    Return the number of ways that mario can traverse the level where mario can either hop by one digit or two digits each turn a level is defined as being an integer where a 1 is something mario can step on and 0 is something mario cannot step on.
    >>> mario_number(10101)
    1
    >>> mario_number(11101)
    2
    >>> mario_number(100101)
    0
    """
    if ____________:
        __________
    elif ____________:
        __________
    else:
        ____________________
```
Solution:

def mario_number(level):
    """
    Return the number of ways that mario can traverse the
    level where mario can either hop by one digit or two
digits each turn a level is defined as being an integer
    where a 1 is something mario can step on and 0 is
    something mario cannot step on.
    >>> mario_number(10101)
    1
    >>> mario_number(11101)
    2
    >>> mario_number(100101)
    0
    """
    if level == 1:
        return 1
    elif level % 10 == 0:
        return 0
    else:
        return mario_number(level // 10) + mario_number((
            level // 10) // 10)
4. Implement the function `make_change`.

```python
def make_change(n):
    """Write a function, make_change that takes in an
    integer amount, n, and returns the minimum number
    of coins we can use to make change for that n,
    using 1-cent, 3-cent, and 4-cent coins.
    Look at the doctests for more examples.
    >>> make_change(5)
    2
    >>> make_change(6) # tricky! Not 4 + 1 + 1 but 3 + 3
    2
    ""
    if ____________:
        return 0
    elif ____________:
        return 1 + make_change(n - 1)
    elif ____________:
        _________________
        __________________
        __________________
        return _________________
    else:
        _________________
        __________________
        __________________
        return _________________
```
Solution:
def make_change(n):
    """Write a function, make_change that takes in an integer amount, n, and returns the minimum number of coins we can use to make change for that n, using 1-cent, 3-cent, and 4-cent coins. Look at the doctests for more examples.
    >>> make_change(5)
    2
    >>> make_change(6) # tricky! Not 4 + 1 + 1 but 3 + 3
    2
    """
    if n < 1:
        return 0
    elif n < 3:
        return 1 + make_change(n - 1)
    elif n < 4:
        use_1 = 1 + make_change(n - 1)
        use_3 = 1 + make_change(n - 3)
        return min(use_1, use_3)
    else:
        use_1 = 1 + make_change(n - 1)
        use_3 = 1 + make_change(n - 3)
        use_4 = 1 + make_change(n - 4)
        return min(use_1, use_3, use_4)