Object-Oriented Programming
We want to write objects that simulate grading in CS10. Fill in the function definitions below to complete our implementation!

```python
import random

class Reader:
    # Every reader has a name and a list of assignment objects they need to grade. The list of assignments should start out empty.
    def __init__(self, name):
        self.name = name
        self.grading_queue = []

    def grade_assignment(self):
        # Assign a random score to the first item in the grading queue and then remove that assignment from the queue
        score = random.randint(0, 10)
        if len(self.grading_queue) > 0:
            self.grading_queue[0].score = score
            self.grading_queue.pop(0)

class Assignment:
    # Every assignment has a student object, assignment title, and score. The score should always start out as 0.
    def __init__(self, student, title):
        self.student = student
        self.title = title
        self.score = 0

class Student:
    # Every student has a name
    def __init__(self, name):
        self.name = name

    def submit(self, assignment, reader):
        # To submit an assignment, add the assignment object to the reader’s grading queue
        reader.grading_queue.append(assignment)
```
Recursion

1. Write a recursive function that takes in a number, \( n \), and determines how many digits it has. Hint: One way to figure out how many digits are in a number is to count how many times you need to divide that number until you get a number less than 10.

```python
def num_digits(n):
    if n < 10:
        return 1
    else:
        return 1 + num_digits(n / 10)
```

2. Write a function called value that takes in a (possibly nested) dictionary and a key in that dictionary, and returns the value of that key.

```python
dict = {'name': 'Pikachu', 'attack': {'move': 'Thunder Shock', 'damage': 40}, 'type': 'electric'}
>>> value(dict, 'damage')
40
```

```python
def value(dict, key):
    if key in dict:
        return dict[key]
    for d in dict.values():
        if value(d, key) is not None:
            return value(d, key)
```

3. You need to buy exactly total pieces of candy, but the grocery stores around you only sell candy in packs of \( x \) and \( y \). Fill out the recursive function `buy_candy` to determine whether you'll be able to successfully buy your candy.

```python
>>> buy_candy(100, 25, 40)
True #25(4) + 0(40) = 100
>>> buy_candy(33, 9, 12)
True #9(1) + 12(2) = 33
>>> buy_candy(10, 4, 8)
False
```

```python
def buy_candy(total, x, y):
    if total == 0:
        return True
    if total < 0:
        return False
    return buy_candy(total - x, x, y) or buy_candy(total - y, x, y)
```