Problem 1. (*Sum of Integers*) Implement the function \_sumOfInts\_ in \_sum\_of\_ints\_\.py that takes an integer \(n\) as argument and returns the sum \(S(n) = 1 + 2 + 3 + \cdots + n\), computed recursively using the recurrence equation

\[
S(n) = \begin{cases} 
1 & \text{if } n = 1, \\
n + S(n - 1) & \text{if } n > 1.
\end{cases}
\]

```bash
$ python3 sum_of_ints.py 100
5050
```

Problem 2. (*Bit Counts*) Implement the functions \_zeros\_ and \_ones\_ in \_bits\_\.py that take a bit string (ie, a string of zeros and ones) \(s\) as argument and return the number of zeros and ones in \(s\), each computed recursively. The number of zeros in a bit string is 1 or 0 (if the first character is '0' or '1') plus the number of zeros in the rest of the string; number of zeros in an empty string is 0 (base case). The number of ones in a bit string can be defined analogously.

```bash
$ python3 bits.py 1010010010011110001011111
zeros = 11, ones = 14, total = 25
```

Problem 3. (*String Reversal*) Implement the function \_reverse\_ in \_reverse\_\.py that takes a string \(s\) as argument and returns the reverse of the string, computed recursively. The reverse of a string is the last character concatenated with the reverse of the string up to the last character; the reverse of an empty string is an empty string (base case).

```bash
$ python3 reverse.py bolton
notlob
```

Problem 4. (*Palindrome*) Implement the function \_isPalindrome\_ in \_palindrome\_\.py, using recursion, such that it returns \texttt{True} if the argument \(s\) is a palindrome (ie, reads the same forwards and backwards), and \texttt{False} otherwise. You may assume that \(s\) is all lower case and doesn’t include any whitespace characters. A string is a palindrome if the first character is the same as the last and the rest of the string is a palindrome; an empty string is a palindrome (base case).

```bash
$ python3 palindrome.py bolton
False
$ python3 palindrome.py madam
True
```

Problem 5. (*Password Checker*) Implement the function \_isValid\_ in \_password\_checker\_\.py that returns \texttt{True} if the given password string meets the following requirements, and \texttt{False} otherwise:

- Is at least eight characters long
- Contains at least one digit (0-9)
- Contains at least one uppercase letter
- Contains at least one lowercase letter
- Contains at least one character that is neither a letter nor a number
Exercise 5 (Recursion and Object-oriented Programming)

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>~/workspace/exercise5</code></td>
<td>File path</td>
</tr>
<tr>
<td><code>$ python3 password_checker.py Abcde1fg</code></td>
<td>Execution of a script</td>
</tr>
<tr>
<td><code>False</code></td>
<td>Output of the script</td>
</tr>
<tr>
<td><code>$ python3 password_checker.py Abcde1@g</code></td>
<td>Execution of a different script</td>
</tr>
<tr>
<td><code>True</code></td>
<td>Output of the different script</td>
</tr>
</tbody>
</table>

Hint: use the `str` methods `isdigit()`, `isupper()`, `islower()`, and `isalnum()`.

**Problem 6. (2D Point)** Define a data type called `Point` in `point.py` that represents a point in 2D. The data type must support the following API:

```python
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def distanceTo(self, q):
        return ((self.x - q.x)**2 + (self.y - q.y)**2)**0.5

    def __str__(self):
        return f'({self.x}, {self.y})'
```

Example:

```
$ python3 point.py 0 1 1 0
p1 = (0.0, 1.0)
p2 = (1.0, 0.0)
d(p1, p2) = 1.4142135623730951
```

**Problem 7. (1D Interval)** Define a data type called `Interval` in `interval.py` that represents a closed 1D interval. The data type must support the following API:

```python
class Interval:
    def __init__(self, lbound, rbound):
        self.lbound = lbound
        self.rbound = rbound

    def __str__(self):
        return f'[{self.lbound}, {self.rbound}]'
```

Example:

```
$ python3 interval.py 3.14
0 1 0.5 1 2 1.6 2.5 2.5 3.5 3.140000 Contains 3.140000
[2.5, 3.5] contains 3.140000
[3.0, 4.0] contains 3.140000
[0.0, 1.0] intersects ([0.5, 1.5], [1.5, 2.0])
[0.0, 1.0] intersects ([1.0, 2.0])
[0.5, 1.6] intersects ([1.5, 2.6])
[1.0, 2.0] intersects ([1.5, 2.6])
[1.5, 2.6] intersects ([2.5, 3.5])
[2.5, 3.5] intersects ([3.0, 4.0])
```

**Problem 8. (Rectangle)** Define a data type called `Rectangle` in `rectangle.py` that represents a rectangle using 1D intervals (i.e., `Interval` objects) to represent its `x` (width) and `y` (height) segments. The data type must support the following API:

```python
class Rectangle:
    def __init__(self, xint, yint):
        self.xint = xint
        self.yint = yint

    def area(self):
        return (self.xint.rbound - self.xint.lbound) * (self.yint.rbound - self.yint.lbound)

    def perimeter(self):
        return 2 * ((self.xint.rbound - self.xint.lbound) + (self.yint.rbound - self.yint.lbound))

    def contains(self, x, y):
        return (self.xint.contains(x) and self.yint.contains(y))

    def intersects(self, r):
        return (self.xint.intersects(r.xint) or self.yint.intersects(r.yint))

    def __str__(self):
        return f'[{self.xint.lbound}, {self.xint.rbound}] x [{self.yint.lbound}, {self.yint.rbound}]'
```

Example:

```
$ python3 rectangle.py 3.14
0 1 0.5 1 2 1.6 2.5 2.5 3.5 3.140000 Contains 3.140000
[2.5, 3.5] contains 3.140000
[3.0, 4.0] contains 3.140000
[0.0, 1.0] intersects ([0.5, 1.5], [1.5, 2.6])
[0.0, 1.0] intersects ([1.0, 2.0])
[0.5, 1.6] intersects ([1.5, 2.6])
[1.0, 2.0] intersects ([1.5, 2.6])
[1.5, 2.6] intersects ([2.5, 3.5])
[2.5, 3.5] intersects ([3.0, 4.0])
```
Exercise 5 (Recursion and Object-oriented Programming)

Files to Submit

1. sum_of ints.py
2. bits.py
3. reverse.py
4. palindrome.py
5. password_checker.py
6. point.py
7. interval.py
8. rectangle.py

Before you submit your files, make sure:

- You do not use concepts from sections beyond “Designing Data Types”.
- Your code is adequately commented, follows good programming principles, and meets any specific requirements such as corner cases and running times.