Can you write this in Python?

```c
#include <stdio.h>
int main(void)
{
    int count;
    for (count = 1; count <= 500; count++)
        printf("I will not throw paper airplanes in class.\n");
    return 0;
}
```
Nested for loop exercise

• Make a table to represent any patterns on each line.

<table>
<thead>
<tr>
<th>line</th>
<th># of dots</th>
<th>(-1 \times \text{line})</th>
<th>(-1 \times \text{line} + 5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>(-1)</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>(-2)</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>(-3)</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>(-4)</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>(-5)</td>
<td>0</td>
</tr>
</tbody>
</table>

• To print a character multiple times, use a for loop.

```python
for j in range(1, 5):
    print("." * j)  # 4 dots
```
Nested for loop solution

• Answer:
  
  ```python
  for line in range(1, 6):
    for j in range(1, (-1 * line + 5 + 1)):
      print('.', end='')
    print(line)
  ```

• Output:
  
  ```plaintext
  ....1
  ...2
  ..3
  .4
  5
  ```
Drawing complex figures

• Use nested `for` loops to produce the following output.

• Why draw ASCII art?
  • Real graphics require a lot of finesse
  • ASCII art has complex patterns
  • Can focus on the algorithms

```
#================#
|      <><>      |
|    <>....<>    |
|  <>........<>  |
|<>............<>|
|<>............<>|
|  <>........<>  |
|    <>....<>    |
|      <><>      |
#================#
```
Development strategy

• Recommendations for managing complexity:

  1. Design the program (think about steps or methods needed).
     • write an English description of steps required
     • use this description to decide the functions
  
    2. Create a table of patterns of characters
     • use table to write your for loops

```
#================#
|      <><>      |
|    <>....<>    |
|  <>........<>  |
|<>............<>|
|<>............<>|
|  <>........<>  |
|    <>....<>    |
|      <><>      |
#================#
```
1. Pseudo-code

- **pseudo-code**: An English description of an algorithm.

- Example: Drawing a 12 wide by 7 tall box of stars

  ```plaintext
  print 12 stars.
  for (each of 5 lines):
    print a star.
    print 10 spaces.
    print a star.
  print 12 stars.
  ```

  `************`

  `*          *`
  `*          *`
  `*          *`
  `*          *`
  `*          *`

  `************`
Pseudo-code algorithm

1. Line
   • #, 16 =, #

2. Top half
   • |
   • spaces (decreasing)
   • <>
   • dots (increasing)
   • <>
   • spaces (same as above)
   • |

3. Bottom half (top half upside-down)

4. Line
   • #, 16 =, #
def main():
    line()
    top_half()
    bottom_half()
    line()

def top_half():
    for line in range(1, 5):
        # contents of each line

def bottom_half():
    for line in range(1, 5):
        # contents of each line

def line():
    # ...
2. Tables

- A table for the top half:
  - Compute spaces and dots expressions from line number

<table>
<thead>
<tr>
<th>line</th>
<th>spaces</th>
<th>line * -2 + 8</th>
<th>dots</th>
<th>4 * line - 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>12</td>
<td>12</td>
</tr>
</tbody>
</table>

#================#
|      <><>      |
|    <>....<>    |
|  <>........<>  |
|<>............<>|
|<>............<>|
|  <>........<>  |
|    <>....<>    |
|      <><>      |
#================#
3. Writing the code

• Useful questions about the top half:
  • Number of (nested) loops per line?
Partial solution

```python
# Prints the expanding pattern of <> for the top half of the figure.
def top_half():
    for line in range(1, 5):
        print("|", end="")
        for space in range(1, line * -2 + 9):
            print(" ", end="")
        print(" <> ", end="")
        for dot in range(1, line * 4 - 3):
            print(".", end="")
        print(" <> ", end="")
        for space in range(1, line * -2 + 8):
            print(" ", end="")
        print("|")
```

Class constants and scope
Scaling the mirror

• Let's modify our Mirror program so that it can scale.
  • The current mirror (left) is at size 4; the right is at size 3.

• We'd like to structure the code so we can scale the figure by changing the code in just one place.

```
#=================================#
|         <><>         |
|       <>....<>       |
|      <>........<>     |
| <>............<>     |
| <>............<>     |
|      <>........<>    |
|       <>....<>       |
|         <><>         |
#=================================#
```

```
#=================#
|     <><>     |
|  <>....<>  |
|<>........<>|
|<>........<>|
|<>........<>|
| <>....<>  |
|     <><>     |
#=================#
```
Constants

- **constant**: A fixed value visible to the whole program.
  - value should only be set only at declaration; shouldn't be reassigned

- **Syntax**:
  - Just like declaring a normal variable:
    ```
    name = value
    ``
  - name is usually in ALL_UPPER_CASE

- **Examples**:
  ```
  DAYS_IN_WEEK = 7
  INTEREST_RATE = 3.5
  SSN = 658234569
  ```
Constants and figures

- Consider the task of drawing the following scalable figure:

```
+--------+
|        |
|        |
|        |
+--------+
```

Multiples of 5 occur many times

```
+--------+
|        |
|        |
+--------+
```

The same figure at size 2
def main():
    draw_line()
    draw_body()
    draw_line()

def draw_line():
    print("+", end="")
    for i in range(1, 11):
        print("/\", end="")
    print("+")

def draw_body():
    for line in range(1, 6):
        print("|", end="")
        for spaces in range(1, 21):
            print(" ", end="")
        print("|")
Adding a constant

```python
HEIGHT = 5
def main():
    draw_line()
    draw_body()
    draw_line()

def draw_line():
    print("+", end="")
    for i in range(1, HEIGHT * 2 + 1):
        print("/\", end="")
    print("+")

def draw_body():
    for line in range(1, HEIGHT + 1):
        print("|", end="")
        for spaces in range(1, HEIGHT * 4 + 1):
            print(" ", end="")
        print("|")
```
Complex figure w/ constant

- Modify the Mirror code to be resizable using a constant.

A mirror of size 4:
```
#=================================#
|      <><>      |
|    <>....<>    |
|  <>........<>  |
|<>............<>|
|<>............<>|
|  <>........<>  |
|    <>....<>    |
|      <><>      |
#=================================#
```

A mirror of size 3:
```
#=================#
|    <><>    |
|  <>....<>  |
|<>........<>|
|<>........<>|
|  <>....<>  |
|    <><>    |
#=================#
Loop tables and constant

- Let's modify our loop table to use `SIZE`
  - This can change the amount added in the loop expression

<table>
<thead>
<tr>
<th>SIZE</th>
<th>line</th>
<th>spaces</th>
<th>dots</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1,2,3,4</td>
<td>6,4,2,0</td>
<td>0,4,8,12</td>
</tr>
<tr>
<td>3</td>
<td>1,2,3</td>
<td>4,2,0</td>
<td>0,4,8</td>
</tr>
</tbody>
</table>

#=================================#
| <><> |
| <><> |
| <><> |
| <><> |
| #=================================#
| #=================================|
SIZE = 4;

# Prints the expanding pattern of <> for the top half of the figure.
def top_half() {
    for line in range(1, SIZE):
        print("|", end="")
        for space in range(1, line * 2 - 2 + (2*SIZE) + 1):
            print(" ", end="")
        print("<>", end="")
        for dot in range(1, line * 4 - 3):
            print(\., end="")
        print("<>", end="")
        for space in range(1, line * 2 - 2 + (2*SIZE) + 1):
            print(" ", end="")
    print("|")
}
Observations about constant

• The constant can change the "intercept" in an expression.
  • Usually the "slope" is unchanged.

SIZE = 4;

for space in range(1, line * -2 + (2 * SIZE)):
    print(" ", end="")

• It doesn't replace every occurrence of the original value.

for dot in range(1, line * 4 - 4 + 1):
    print(".", end="")