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;
}
```
**Review: for loops**

**loop**: Repeat one or more statements a specified number of times

```python
for i in range(1, 6):
    print(i*i)  # square variable i
```

• Output:

1  
4  
9  
16  
25  

• The loop repeats 5 times.
Review: print conventions

• `print(' ', end='')`

• **Adding `, end=''** allows you to print without moving to the next line
  
  • allows you to print on the same line; no advancing to the next line
  • the quotes contain any valid string

  ```python
  for i in range(-3, 3):
    print(i, end=' ')
  ```

• **Output:**
  
  ```
  -3 -2 -1 0 1 2
  ```
New: Changing step size

• Add a third number to the end of range, this is the step size
  • A negative number will count down instead of up

```python
for i in range(10, 0, -1):
    print(i, end=' ')
```

• output:

```
10 9 8 7 6 5 4 3 2 1
```

• How would we produce the following Rocket Countdown?

```
T-minus 10! 9! 8! 7! 6! 5! 4! 3! 2! 1! blastoff!!
The end.
```
Rocket Countdown

• Use a negative number for step size
• Use `str()` and concatenation in `print()`

```python
print("T-minus ")
for i in range(10, 0, -1):
    print(str(i) + '! ', end='')
print("blastoff!!")
print("The end.")
```

• Output:

```
T-minus 10! 9! 8! 7! 6! 5! 4! 3! 2! 1! blastoff!!
The end.
```
Nested loops

- **nested loop**: A loop placed inside another loop.

```python
for i in range(1, 6):
    for j in range(1, 11):
        print("*", end="")
    print()  # to end the line
```

- **Output**:

  **********
  **********
  **********
  **********
  **********

- The outer loop repeats 5 times; the inner one 10 times.
  - "sets and reps" exercise analogy
Nested for loop exercise

• What is the output of the following nested for loops?

```python
for i in range(1, 6):
    for j in range(1, i + 1):
        print("*", end='')
    print()
```

• Output:

```
*  
** 
*** 
**** 
*****
```
Nested for loop exercise

• What is the output of the following nested for loops?

```python
for i in range(1, 6):
    for j in range(1, i + 1):
        print(i, end='')
    print()
```

• Output:

```
1
22
333
4444
55555
```
Complex lines

• What nested `for` loops produce the following output?

```
1
2
3
4
5
```

- `inner loop (repeated characters on each line)`

```
...1
...2
..3
.4
5
```

- `outer loop (loops 5 times because there are 5 lines)`

• We must build multiple complex lines of output using:
  - an `outer "vertical" loop` for each of the lines
  - `inner "horizontal" loop(s)` for the patterns within each line
Outer and inner loop

• First write the outer loop, from 1 to the number of lines.

```python
for line in range(1, 6):
    ...
```

• Now look at the line contents. Each line has a pattern:
  • Zero or more dots, then a number

```plaintext
....1
...2
..3
.4
5
```

• Observation: the number of dots is related to the line number.
Mapping loops to numbers

for count in range(1, 6):
    print( ... )

• What statement in the body would cause the loop to print:
  4 7 10 13 16

for count in range(1, 6):
    print(3 * count + 1, end=' ');
Loop tables

for count in range(1, 6):
    print(…)

• What statement in the body would cause the loop to print:
  2  7  12  17  22

• To see patterns, make a table of count and the numbers.
  • Each time count goes up by 1, the number should go up by 5.
  • But count * 5 is too great by 3, so we subtract 3.

<table>
<thead>
<tr>
<th>count</th>
<th>number to print</th>
<th>5 * count</th>
<th>5 * count - 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>15</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>17</td>
<td>20</td>
<td>17</td>
</tr>
<tr>
<td>5</td>
<td>22</td>
<td>25</td>
<td>22</td>
</tr>
</tbody>
</table>
Loop tables question

• What statement in the body would cause the loop to print:
  17 13 9 5 1

• Let's create the loop table together.
  • Each time count goes up 1, the number printed should ...
  • But this multiple is off by a margin of ...

<table>
<thead>
<tr>
<th>count</th>
<th>number to print</th>
<th>(-4 \times \text{count})</th>
<th>(-4 \times \text{count} + 21)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>17</td>
<td>-4</td>
<td>17</td>
</tr>
<tr>
<td>2</td>
<td>13</td>
<td>-8</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>-12</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>-16</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>-20</td>
<td>1</td>
</tr>
</tbody>
</table>
Another view: Slope-intercept

• The next three slides present the mathematical basis for the loop tables. Feel free to skip it.

<table>
<thead>
<tr>
<th>count (x)</th>
<th>number to print (y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>17</td>
</tr>
<tr>
<td>5</td>
<td>22</td>
</tr>
</tbody>
</table>
Another view: Slope-intercept

- **Caution**: This is algebra, not assignment!

- Recall: slope-intercept form ($y = mx + b$)

- Slope is defined as “rise over run” (i.e., $\text{rise} / \text{run}$). Since the “run” is always 1 (we increment along $x$ by 1), we just need to look at the “rise”. The rise is the difference between the $y$ values. Thus, the slope ($m$) is the difference between $y$ values; in this case, it is $+5$.

- To compute the $y$-intercept ($b$), plug in the value of $y$ at $x = 1$ and solve for $b$. In this case, $y = 2$.

  \[
  \begin{align*}
  y &= m \times x + b \\
  2 &= 5 \times 1 + b \\
  \text{Then } b &= -3
  \end{align*}
  \]

- So the equation is

  \[
  \begin{align*}
  y &= m \times x + b \\
  y &= 5 \times x - 3 \\
  y &= 5 \times \text{count} - 3
  \end{align*}
  \]

- $\text{count (x)}$ | $\text{number to print (y)}$
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>17</td>
</tr>
<tr>
<td>5</td>
<td>22</td>
</tr>
</tbody>
</table>
Another view: Slope-intercept

• Algebraically, if we always take the value of \( y \) at \( x = 1 \), then we can solve for \( b \) as follows:

\[
\begin{align*}
  y &= m \times x + b \\
  y_1 &= m \times 1 + b \\
  y_1 &= m + b \\
  b &= y_1 - m
\end{align*}
\]

• In other words, to get the \( y \)-intercept, just subtract the slope from the first \( y \) value (\( b = 2 - 5 = -3 \))

• This gets us the equation

\[
\begin{align*}
  y &= m \times x + b \\
  y &= 5 \times x - 3 \\
  y &= 5 \times \text{count} - 3
\end{align*}
\]

(which is exactly the equation from the previous slides)
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 * line</th>
<th>-1 * 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(".") # 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:
  ```
  ....1
  ...2
  ..3
  .4
  5
  ```
Nested for loop exercise

• What is the output of the following nested for loops?

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

• Answer:

```
....1
...22
..333
 .4444
 55555
```
Nested for loop exercise

• Modify the previous code to produce this output:
  ....1
  ...2.
  ..3..
  .4...
  5....

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

• Use nested for loops to produce the following output.

• Why draw ASCII art?
  • Real graphics are quite intricate
  • ASCII art has complex patterns
  • Can focus on the algorithms

---

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

---
Development strategy

• Recommendations for managing complexity:

  1. Design the program (think about steps or functions 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. Pseudocode

- **pseudocode**: An English description of an algorithm.

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

  ```plaintext
pseudocode:

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

  ********************************************
  *          *
  *          *
  *          *
  *          *
  *          *
  ********************************************
  ```
Pseudocode 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 =, #
Functions from pseudocode

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

# 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("|")
# 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("|")