Need:
- a random 2D grid

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>3</td>
<td>7</td>
<td>2</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>7</td>
<td>2</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>8</td>
<td>7</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>6</td>
<td>4</td>
<td>9</td>
<td>3</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>9</td>
<td>3</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>2</td>
<td>5</td>
<td>8</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>3</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>1</td>
<td>9</td>
<td>7</td>
<td>8</td>
<td>6</td>
<td>5</td>
</tr>
</tbody>
</table>
Need:
- a random 2D grid
- to mark cells
Don’t care about a cell’s value after it has been filled.
So use any value that can’t ever be a real cell value to mark filled cells.
Note: we’re using integers as:
Note: we’re using integers as:

- Actual data values
Note: we’re using integers as:

- Actual data values

- Flags to represent cell states
Note: we’re using integers as:
- Actual data values
- Flags to represent cell states

It’s simple to do...
Note: we’re using integers as:

- Actual data values
- Flags to represent cell states

It’s simple to do...

...but if we ever get data whose values are those we’ve been using as flags, our program will interpret them as flags.
Note: we’re using integers as:
- Actual data values
- Flags to represent cell states
It’s simple to do...
...but if we ever get data whose values are those we’ve been using as flags, our program will interpret them as flags

This kind of error can be very hard to track down
Are grids always square?

```
<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>3</td>
<td>7</td>
<td>2</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>7</td>
<td>2</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>8</td>
<td>7</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>6</td>
<td>4</td>
<td>9</td>
<td>3</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>-1</td>
<td>9</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>-1</td>
<td>-1</td>
<td>6</td>
<td>8</td>
</tr>
</tbody>
</table>
```
Are grids always square?

Are they always odd × odd (so that there is a unique center square)?

<table>
<thead>
<tr>
<th></th>
<th>5</th>
<th>3</th>
<th>7</th>
<th>2</th>
<th>6</th>
<th>1</th>
<th>1</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>7</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>8</td>
<td>7</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>6</td>
<td>4</td>
<td>9</td>
<td>3</td>
<td>9</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>-1</td>
<td>9</td>
<td>7</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>-1</td>
<td>-1</td>
<td>6</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>
How general should we make the first version of our program?

<table>
<thead>
<tr>
<th></th>
<th>5</th>
<th>3</th>
<th>7</th>
<th>2</th>
<th>6</th>
<th>1</th>
<th>1</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>7</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>8</td>
<td>7</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>6</td>
<td>4</td>
<td>9</td>
<td>3</td>
<td>9</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>-1</td>
<td>9</td>
<td>7</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>-1</td>
<td>-1</td>
<td>6</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>
How general should we make the first version of our program?

“Don’t build it until you need it.”
How general should we make the first version of our program?

<table>
<thead>
<tr>
<th>5</th>
<th>3</th>
<th>7</th>
<th>2</th>
<th>6</th>
<th>1</th>
<th>1</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>7</td>
<td>2</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>8</td>
<td>7</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>6</td>
<td>4</td>
<td>9</td>
<td>3</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>-1</td>
<td>9</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>-1</td>
<td>-1</td>
<td>6</td>
<td>8</td>
</tr>
</tbody>
</table>

“Don’t build it until you need it.”

vs.

“A week of hard work can sometimes save you an hour of thought.”
Like many generalizations, these rules are:
Like many generalizations, these rules are:

- True
Like many generalizations, these rules are:

- True

- Not particularly useful
Like many generalizations, these rules are:
- True
- Not particularly useful

Knowing what rules to apply when comes with experience
Like many generalizations, these rules are:

- True
- Not particularly useful

Knowing what rules to apply when comes with experience

The only way to get experience is to work through many examples
Problem: Python doesn’t actually have 2D arrays
But it does have 1D lists
But it does have 1D lists which can refer to other lists.
But it does have 1D lists
Which can refer to other lists
This gives us double subscripts
But it does have 1D lists. Which can refer to other lists. This gives us double subscripts...which is really what we mean by “two-dimensional”
# Create an NxN grid of random integers in 1..Z.
assert N > 0, "Grid size must be positive"
assert N%2 == 1, "Grid size must be odd"

grid = []
for x in range(N):
    grid.append([])
    for y in range(N):
        grid[-1].append(1)  # FIXME: need a random value
# Create an N\times N grid of random integers in 1..Z.
assert N > 0, "Grid size must be positive"
assert N%2 == 1, "Grid size must be odd"

grid = []
for x in range(N):
    grid.append([])
for y in range(N):
    grid[-1].append(1)  # FIXME: need a random value
# Create an N x N grid of random integers in 1..Z.
assert N > 0, "Grid size must be positive"
assert N%2 == 1, "Grid size must be odd"

grid = []
for x in range(N):
    grid.append([])
    for y in range(N):
        grid[-1].append(1)  # FIXME: need a random value

grid ----> □
# Create an NxN grid of random integers in 1..Z.
assert N > 0, "Grid size must be positive"
assert N%2 == 1, "Grid size must be odd"
grid = []
for x in range(N):
    grid.append([])
    for y in range(N):
        grid[-1].append(1)  # FIXME: need a random value

grid

-->

Invasion Percolation

The Grid
# Create an N x N grid of random integers in 1..Z.
assert N > 0, "Grid size must be positive"
assert N%2 == 1, "Grid size must be odd"
grid = []
for x in range(N):
    grid.append([])
    for y in range(N):
        grid[-1].append(1) # FIXME: need a random value
# Create an NxN grid of random integers in 1..Z.
assert N > 0, "Grid size must be positive"
assert N%2 == 1, "Grid size must be odd"
grid = []
for x in range(N):
    grid.append([0])
    for y in range(N):
        grid[-1].append(1) # FIXME: need a random value

grid
created by

Greg Wilson

May 2010