Week 5

numpy arrays

STAT 198/298 Fall 2020

Send your phone here

Or send a browser to `slido.com`, event `#Z837`.

Agenda

1. Review: Drawing the Polyforce
2. Why we need NumPy

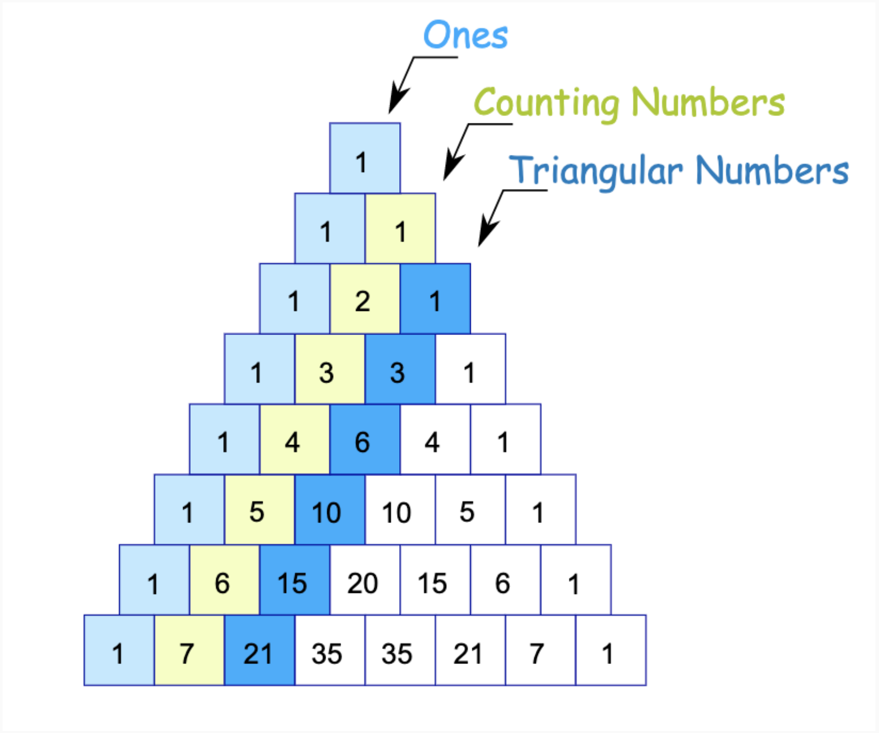
```python
def draw_polyforce(n, full_side_length = 200, max_n = 55):
    if n > max_n:
        raise ValueError("n must not exceed", max_n)
    triangular_numbers = [0]
    i = 0
    while triangular_numbers[-1] < n:
        triangular_numbers.append(triangular_numbers[i] + i)
        i += 1
    if n not in triangular_numbers:
        raise ValueError("n must be a triangular number")
    n_rows = len(triangular_numbers) - 2
    h = math.sqrt(3)/2 * full_side_length
    x_coords = [x/n_rows * full_side_length for x in
                range(0, n_rows)]
    y_coords = [x/n_rows * h - h/2 for x in range(0, n_rows)
    t.penup()
    for i in range(len(y_coords)):
        for j in range(len(x_coords)):
            t.goto(x_coords[j], y_coords[i])
            draw_golden_tri(full_side_length/n_rows)
        x_coords = [(a + b) / 2 for a, b in
                    zip(x_coords[1:], x_coords[:-1])]
```

Check 1: n too large

```
def draw_polyforce(n, full_side_length = 200, max_n = 55):
    if n > max_n:
        raise ValueError("n must not exceed", max_n)
```

Check 2: n is a triangular number

Pascal's Triangle



Let's build a list of triangular numbers through iteration.

`for` loop

```python
triangular_numbers = [0]
for i in range(5):
    triangular_numbers.append(triangular_numbers[i] + i)
print(triangular_numbers)
```

```
## [0, 0, 1, 3, 6, 10]
```

Downside: have to fix the length of the list.

while loop

```
n = 10
triangular_numbers = [0]
i = 0
while triangular_numbers[-1] < n:
    triangular_numbers.append(triangular_numbers[i] + i)
    i += 1
```

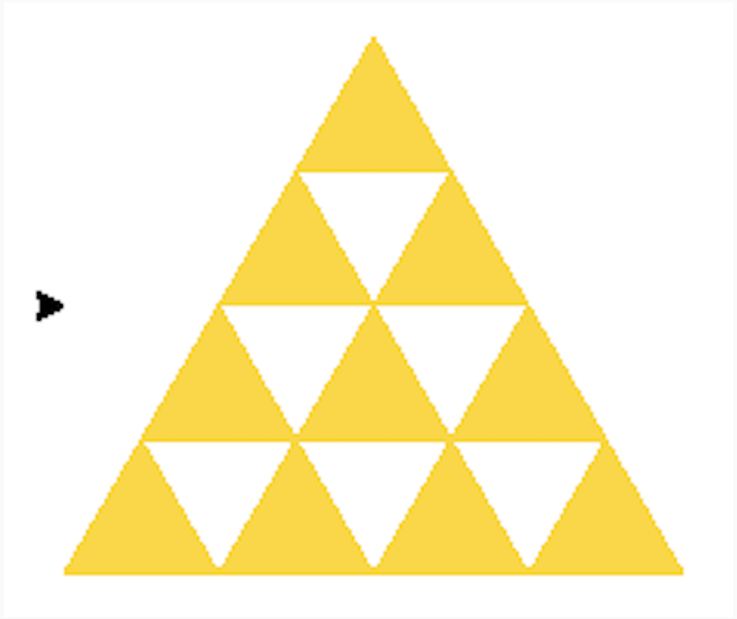
```
## [0, 0, 1, 3, 6, 10]
```

Check 2: n is a triangular number

```
if n not in triangular_numbers:
    raise ValueError("n must be a triangular number")
```

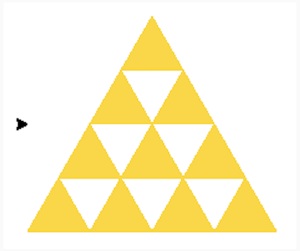
Moving the Turtle

The goal:



How do we direct the turtle where to draw each triangle?

Moving the Turtle



One approach:

- Create list of x-coordinates of lower left of each triangle
- Create corresponding list of y-coordinates
- Iterate through lists and call `draw_golden_tri()` in each loop

List operations in Python

```
full_side_length = 200
n_rows = len(triangular_numbers) - 2
x_coords = [x/n_rows * full_side_length for x in
            range(0, n_rows)]
```

```
## [0.0, 50.0, 100.0, 150.0]
```

In R

```
x_coords = 1:5 / n_rows * full_side_length
```

Observation:

- Even simple operations on the elements of a python list require iterations: loops or list comprehension.
- This is syntactically tortured and slow.

NumPy

- *Numerical Python* (2005)
- Makes available the *numpy array*, efficient storage and operations on arrays
- Very natural for R users: np.array ≈ R vector (matrix/array)

Creating an array

From a list

```
import numpy as np
np.array([1, 5, 3])
```

```
## array([1, 5, 3])
```

From scratch

```
np.zeros(3)
```

```
## array([0., 0., 0.])
```

Also: `np.ones()`, `np.full()`, `np.eye()`, ...

Subsetting

Same as a list.

```
a = np.array([1, 5, 3])
a[0]
```

```
## 1
```

```
a[-1]
```

```
## 3
```

```
a[0:2]
```

```
## array([1, 5])
```

Logical Subsetting with lists

Question: How do I extract the values less than 4?

```
l = [1, 5, 3]
```

```
[x for x in l if x < 4]
```

```
## [1, 3]
```

```
emo::ji("vomit")
```

```
## 🤢
```

Logical Subsetting with arrays

Questions: How do I extract the values less than 4?

```
a = np.array([1, 5, 3])
```

```
a[a < 4]
```

```
## array([1, 3])
```

```
emo::ji("grin")
```

```
## 😁
```

```
a < 4
```

```
## array([ True, False,  True])
```

What is its type?

`CODE`

```
type(a)
```

```
## <class 'numpy.ndarray'>
```

What are its attributes and methods? (`dir(a)`)

```
a.shape
```

```
## (3,)
```

```
a.size
```

```
## 3
```

Lists vs arrays

Lists

```
l1 = [1, 3, 5]
l2 = [4, 5, 6]
print(l1 + l2)
```

```
## [1, 3, 5, 4, 5, 6]
```

concatenation

Arrays

```
a1 = np.array(l1)
a2 = np.array(l2)
a1 + a2
```

```
## [ 5  8 11]
```

element-wise addition

Polyforce revisited

List operations (iterated)

```
x_coords = [x/n_rows * full_side_length for x in
               range(0, n_rows)]
print(x_coords)
```

```
## [0.0, 50.0, 100.0, 150.0]
```

Array operations (element-wise)

```
x_coords = np.array(range(0, n_rows))/n_rows * full_side_len
print(x_coords)
```

```
## [  0.   50. 100. 150.]
```

Array as a data structure

It is:

1. Ordered
2. Mutable
3. Homogeneous

Homogeneous arrays

Question: What is the output of the following code?

```
a = np.array([3.14, "hello", True])
```

```
## ['3.14' 'hello' 'True']
```

Question: What is the output of the following code?

```
a = np.array([3.14, 3])
```

Arrays will automatically *upcast/coerce* to make a homogeneous structure.

Creating 2D arrays

From a list of lists

```
a2 = np.array([[1, 5, 3],
               [4, 5, 6]])
print(a2)
```

```
## [[1 5 3]
##  [4 5 6]]
```

```
a2.shape
```

```
## (2, 3)
```

```
a2.size
```

```
## 6
```

Creating 2D arrays

From scratch

```
a2 = np.ones((2, 3))
print(a2)
```

```
## [[1. 1. 1.]
##  [1. 1. 1.]]
```

```
a2.shape
```

```
## (2, 3)
```

```
a2.size
```

```
## 6
```

Subsetting 2D arrays

Question: How do I extract the 4?

```
a2 = np.array([[1, 5, 3],
               [4, 5, 6]])
```

```
a2[1]
```

```
## array([4, 5, 6])
```

```
a2[1][0]
```

```
## 4
```

```
a2[1, 0]
```

```
## 4
```

Subsetting 2D arrays

Question: How do I extract the subarray contain the left two columns?

```
a2 = np.array([[1, 5, 3],
               [4, 5, 6]])
```

```
a2[:, 0:2]
```

```
## array([[1, 5],
##        [4, 5]])
```