Strings

- A string is a list(array) of characters. Similar to char, you can use either " " or ' ' to show a string.
- Python has a set of built-in methods that you can use on strings.

1. Given a non-empty string, create pyramid pattern as in the following example:
   "xyz" =>

```
....x....
..y.x.y..
z.y.x.y.z
```

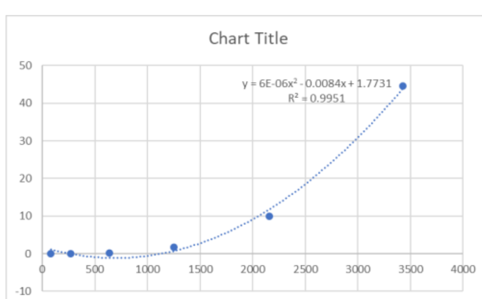
- Here, we have two helpful methods that can help us: join and center.
- Length of each row is the same and equal to $2(2n-1)-1 = 4n-3$.
- We will come up with the following algorithm for this problem:

def **pyramid**(x: str)
$st = ""$
$n = len(x)$
for i **in range** $(0, n)$
    $temp1 = x[0:i+1]$
    $temp2 = x[i:0:-1]$
    $temp = '.'.join(temp2 + temp1)$
    $st = st + temp.center(4*n-3, ".")$
    if $(i \mathrel{!}= n-1)$
        $st = st + "\backslash n"$
return st

Running Time and Time Complexity

- To analyze the efficacy of an algorithm, we usually consider two measurements: time and memory needed.
- An analysis of the time required to solve a problem of a particular size involves the ***time complexity*** of the algorithm. An analysis of the computer memory required involves the ***space complexity*** of the algorithm.

2. Let $T(n)$ be the exact running time of the above algorithm with input size n on my laptop, use an experiment to find $T(n)$.



- The **time complexity** of an algorithm can be expressed in terms of the number of operations used by the algorithm when the input has a particular size.
- We only care about the number of operations, because each operation might have different running times on different machines.

3. What is the time complexity of the algorithm in question 1?

def **pyramid**(x: str)

1	$st\ =\ ""$	1 operation
2	$n\ =\ len(x)$	1 operation
3	**for** i **in range** $(0, n)$	line 4 -9 runs n times
4	$temp1\ =\ x[0 : i+1]$	i operations, $i = 1 \ldots n$
5	$temp2\ =\ x[i : 0 : -1]$	$i-1$ operations, $i = 1 \ldots n$
6	$temp\ =\ '.'.join(temp2\ +\ temp1)$	$(2i-1) \times 2 - 1$ operations
7	$st\ =\ st\ +\ temp.center(4 * n - 3, ".")$	$4n - 3$ operations
8	**if** $(i\ !=\ n-1)$	1 operation
9	$st\ =\ st\ +\ "\backslash n"$	1 operation except for the last iteration
10	**return** st	1 operation

In total, there are $2 + \sum_{i=1}^{n}(6i-4) + (4n-3+2)$

$$= 2 + n(4n-1) + \sum_{i=1}^{n}(6i-4) = (2 + 4n^2 - n) + \frac{(2+6n-4)n}{2} = 7n^2 - 2n + 2$$

Growth of Functions

- [Big-Oh Notation] Let f and g be functions from the set of integers or the set of real numbers to the set of real numbers. We say that $f(x)$ is $O(g(x))$ if there are constants C and k such that:

$$|f\ (x)| \leq\ C|g(x)|$$

whenever $x\ >\ k$. This is read as "$f\ (x)$ is big – oh of $g(x)$."
  - In other words, when x is large enough, $f(x)$ is upper bounded by a constant time $g(x)$.
  - Big-Oh notation is called the ***asymptotic upper bound*** of a function.

4. Show that $f(n) = 7n^2 - 2n + 2$ is $O(n^2)$.

   Solution: We can choose $k = 1$ and $C = 20$ (there are infinite choices of k and C). When $n > 1$, we have $7n^2 - 2n + 2 \leq 20\,n^2$.

- In the above example, we also have that $n^2 \leq 7n^2 - 2n + 2$, for $n > 1$. Thus, we also have $g(n) = n^2$ is $O(f(n))$.
- If $f(x)$ is $O(g(x))$ and $g(x)$ is $O(f(x))$, we say that $f(x)$ and $g(x)$ are of the same order.

5. Show that $7n^2 - 2n + 2$ is $O(n^3)$.
   Solution: when $n > 7$, we have $n^3 > 7n^2 - 2n + 2$ (here I chose $k = 7, C = 1$ in the definition).