Asymptotic notation

- Need a formal way to talk about relationship between running time and size of inputs
- Mostly interested in what happens as size of inputs gets very large, i.e. approaches infinity

Example

```
def f(x):
    for i in range(1000):
        ans = i
    for i in range(x):
        ans += 1
    for i in range(x):
        for j in range(x):
            ans += 1
```

Complexity is $1000 + 2x + 2x^2$, if each line takes one step

Example

- $1000 + 2x + 2x^2$
- If x is small, constant term dominates
  - E.g., x = 10 then 1000 of 1220 steps are in first loop
- If x is large, quadratic term dominates
  - E.g. x = 1,000,000, then first loop takes 0.000000005% of time, second loop takes 0.0001% of time (out of 2,000,002,001,000 steps)!

Example

- So really only need to consider the nested loops (quadratic component)
- Does it matter that this part takes $2x^2$ steps, as opposed to say x^2 steps?
  - For our example, if our computer executes 100 million steps per second, difference is 5.5 hours versus 2.25 hours
  - On the other hand if we can find a linear algorithm, this would run in a fraction of a second
  - So multiplicative factors probably not crucial, but order of growth is crucial

Rules of thumb for complexity

- Asymptotic complexity
  - Describe running time in terms of number of basic steps
  - If running time is sum of multiple terms, keep one with the largest growth rate

    Problem size $\to \infty$
  - If remaining term is a product, drop any multiplicative constants
- Use "Big O" notation (aka Omicron)
  - Gives an upper bound on asymptotic growth of a function