

# Lecture Notes on Orders of Convergence

by  
Kenny Erleben  
2016

# Applying Numerical Methods

- Numerical methods are often iterative in nature. They take one computational step that yields the next iterate
- We wish to know if the iterations converge to a solution
- We wish to know how many steps we need to take to obtain a desired accuracy
- We wish to know how expensive the computational step is

# Applying Numerical Methods

- Convergence theorems are going to give us “conditions” under which we know we will converge to a solution
- Convergence theorems are going to tell us the “rate” at which we will converge (how fast the error drops to zero). We will study the definitions of order of convergence rates to know what this means
- To study how errors behave little “o” and big “O” are tools often being used. Big “O” is also often used to analyse how expensive a computational step is

# Applying Numerical Methods

- For now we have not yet “really” implemented any iterative numerical methods for solving any “problems”. However, we can imagine the iterative nature of such methods, and for each step the methods take we can imagine computing a measure of the current error (a real number).
- As computation goes on the iterative numerical method will create a sequence of such error values
- Hence, our learning will begin with infinite sequences of numbers.

# Infinite Sequence of Numbers

$$\{x_n\} = \{x_0, x_1, x_2, x_3, \dots\}$$

# A Simple Example

$$x_n = \frac{1}{n}, \quad \forall n > 0$$

# Variation in Notation

$$x_n = \frac{1}{n}, \quad n \geq 1$$

$$\{x_n\} = \left\{ \frac{1}{n} \right\}, \quad n \geq 1$$

$$\{x_n\}_1^K = \left\{ \frac{1}{n} \right\}_1^K, \quad K \geq 1$$

# Analysing a Sequence of Numbers

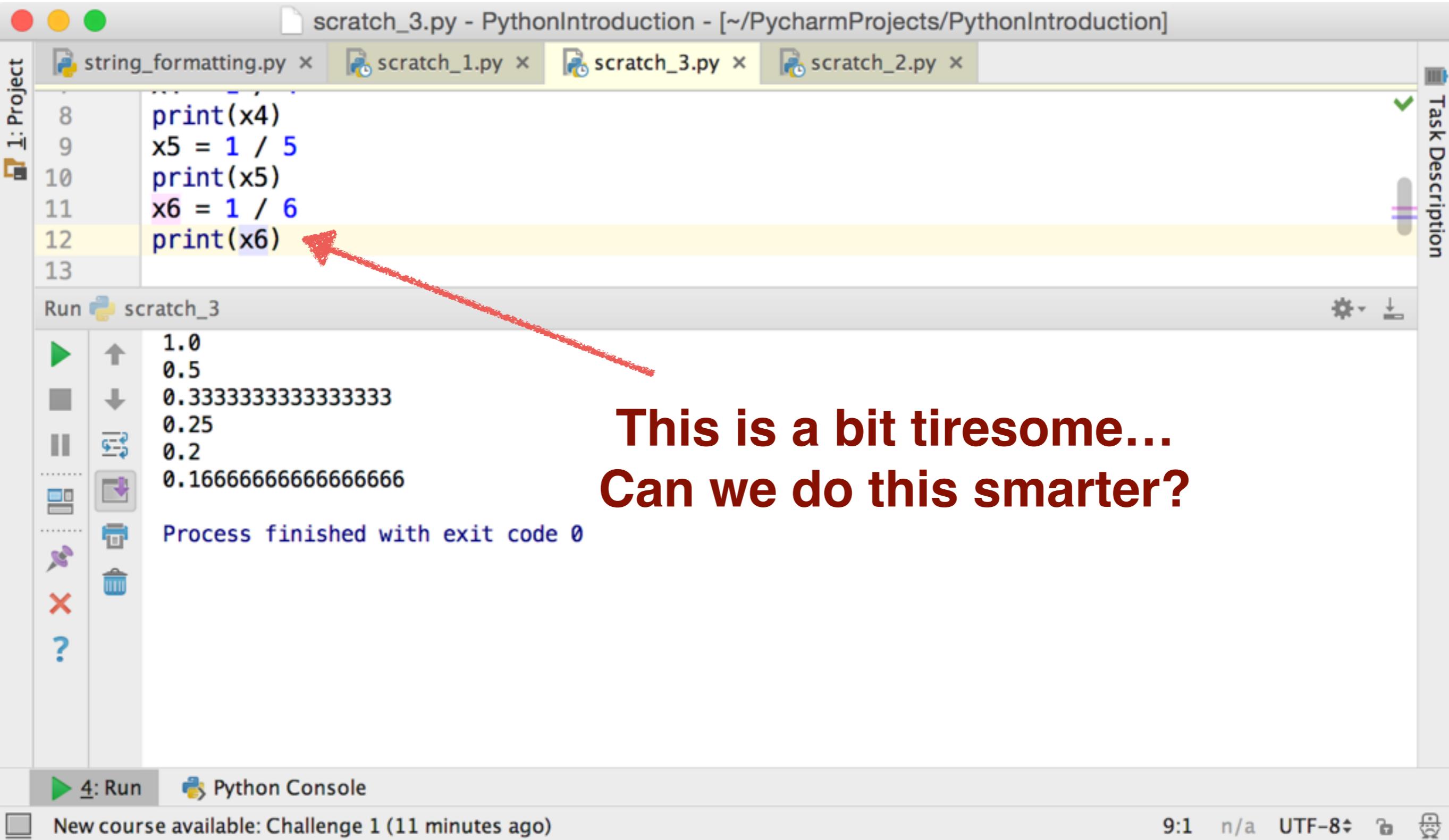
$$\lim_{n \rightarrow \infty} x_n = x^*$$

First Question: Does the sequence of numbers converge or not?

# A Graphically Experimental Approach

Use a computer to compute and plot numbers  
to get an idea of the tendency

# Sequence of Numbers in Python



A screenshot of the PyCharm IDE interface. The title bar shows "scratch\_3.py - PythonIntroduction - [~/PycharmProjects/PythonIntroduction]". The tab bar has four tabs: "string\_formatting.py", "scratch\_1.py", "scratch\_3.py" (which is selected), and "scratch\_2.py". The code editor shows the following Python code:

```
8     print(x4)
9     x5 = 1 / 5
10    print(x5)
11    x6 = 1 / 6
12    print(x6)
13
```

A red arrow points from the text "This is a bit tiresome..." to the line "print(x6)" in the code editor.

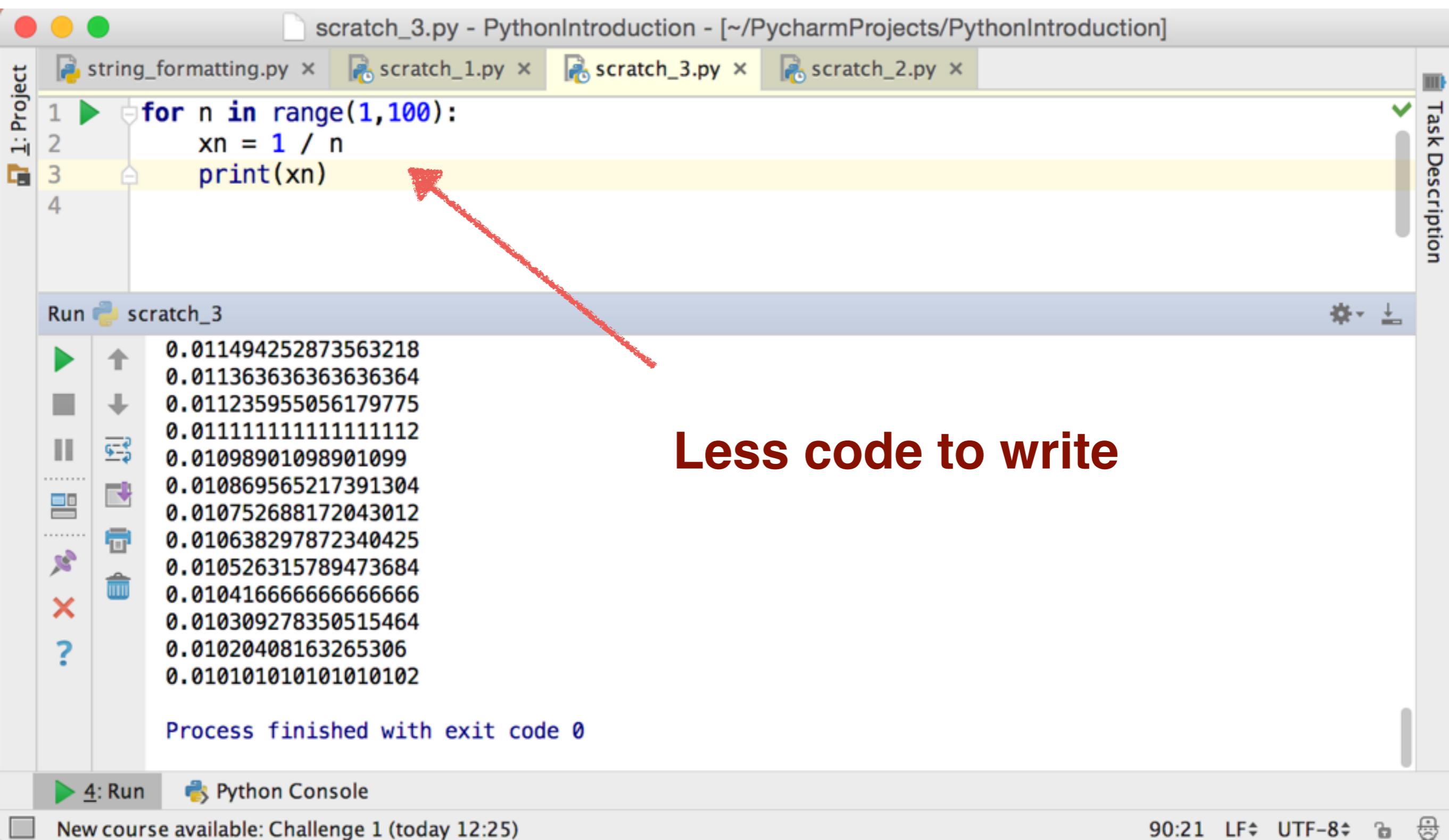
The run tool window below shows the output of the code:

```
Run scratch_3
1.0
0.5
0.3333333333333333
0.25
0.2
0.1666666666666666
Process finished with exit code 0
```

**This is a bit tiresome...  
Can we do this smarter?**

At the bottom, there are buttons for "4: Run" and "Python Console". The status bar at the bottom right shows "9:1 n/a UTF-8" and other icons.

# Sequence of Numbers in Python



A screenshot of the PyCharm IDE interface. The top bar shows the title "scratch\_3.py - PythonIntroduction - [~/PycharmProjects/PythonIntroduction]". Below the title, there are tabs for "string\_formatting.py", "scratch\_1.py", "scratch\_3.py" (which is selected), and "scratch\_2.py". The main code editor window contains the following Python code:

```
1 for n in range(1,100):
2     xn = 1 / n
3     print(xn)
4
```

The third line, "print(xn)", is highlighted with a yellow background and has a red arrow pointing from it towards the output window. The output window below shows the results of the execution:

```
Run scratch_3
0.011494252873563218
0.011363636363636364
0.011235955056179775
0.01111111111111112
0.01098901098901099
0.010869565217391304
0.010752688172043012
0.010638297872340425
0.010526315789473684
0.010416666666666666
0.010309278350515464
0.01020408163265306
0.010101010101010102

Process finished with exit code 0
```

On the left side of the interface, there is a vertical toolbar with various icons for file operations like opening, saving, and deleting files.

**Less code to write**

# Sequence of Numbers in Python

```
scratch_3.py - PythonIntroduction - [~/PycharmProjects/PythonIntroduction]
string_formatting.py x scratch_1.py x scratch_3.py x scratch_2.py x
1: Project
1 x = [ 1 / n for n in range(1,100)]
2
3 for xn in x:
4     print(xn)
5
Run scratch_3
0.011494252873563218
0.011363636363636364
0.011235955056179775
0.01111111111111112
0.01098901098901099
0.010869565217391304
0.010752688172043012
0.010638297872340425
0.010526315789473684
0.010416666666666666
0.010309278350515464
0.01020408163265306
0.010101010101010102
Process finished with exit code 0
```

Even more compact

This output is not so easy to read

# Plotting Sequence of Numbers in Python

scrach\_3.py - PythonIntroduction - [~/PycharmProjects/PythonIntroduction]

string\_formatting.py x scratch\_1.py x scratch\_3.py x scratch\_2.py x

1: Project

1 import matplotlib.pyplot as plt  
2  
3 x = [1 / n for n in range(1, 100)]  
4  
5 plt.plot(x)  
6 plt.show()

Run scratch\_3

Task Description

Let us plot output instead

Figure 1

!!!Looks like it is converging towards zero!!!

4: Run Python Console

New course available: Challenge 1 (41 minutes ago)

8

# Plotting Sequence of Numbers in Python

scratch\_3.py - PythonIntroduction - [~/PycharmProjects/PythonIntroduction]

string\_formatting.py x scratch\_1.py x scratch\_3.py x scratch\_2.py x

1: Project

```
1 import matplotlib.pyplot as plt
2
3 x = [1/n for n in range(1,100)]
4
5 plt.plot(x,'r+')
6 plt.ylabel('value')
7 plt.xlabel('n')
8 plt.show()
```

Sprinkles on top

Run: scratch\_3 scratch\_3 scratch\_3

Process finished with exit code 0

Figure 1

4: Run Python Console

New course available: Challenge 1 (today 12:25)

3:1 LF UTF-8

Task Description

# Formal Proofs

---

## We need Limits

We write

$$\lim_{n \rightarrow \infty} x_n = c$$

If and only if

$$\forall \varepsilon > 0, \exists N > 0$$

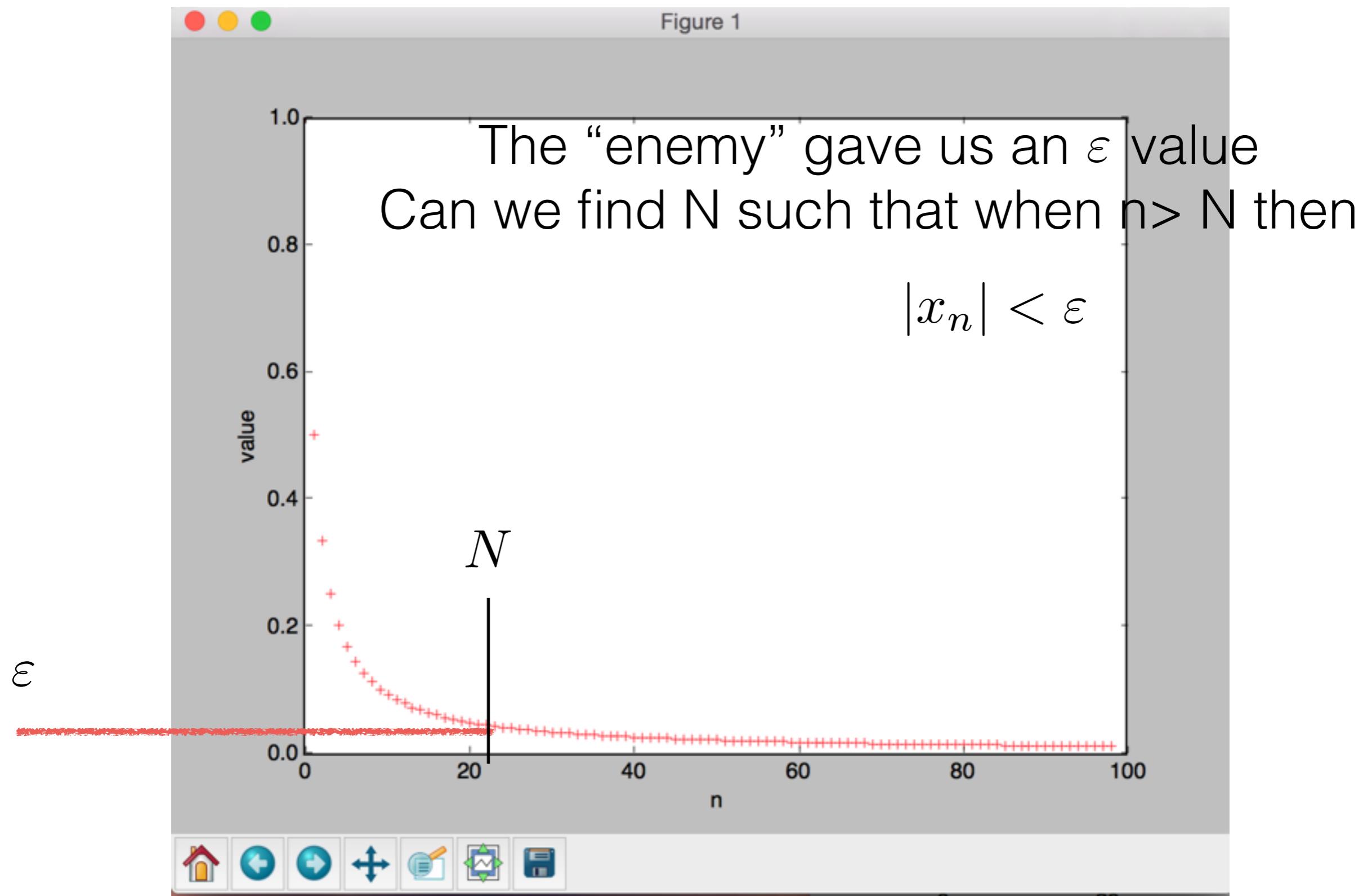
such that

$$n > N \Rightarrow |x_n - c| < \varepsilon$$

### Recipe

- 1) Somebody gives you an epsilon value
- 2) You prove you can find N-value so the inequality is true

# Idea for Proof



# Formal Proof for Simple Example

Prove

$$\lim_{n \rightarrow \infty} \frac{1}{n} = 0$$

Given any  $\varepsilon > 0$  and  $n > N$  then

$$\left| \frac{1}{n} \right| = \frac{1}{n} < \frac{1}{N}$$

Guess a N-value

$$N > \frac{1}{\varepsilon}$$

Verify inequality from formal definition

$$\left| \frac{1}{n} \right| < \frac{1}{N} < \frac{1}{\frac{1}{\varepsilon}} = \varepsilon$$

“True” as wanted so our guess proved existence of N-value

# Reflections on usefulness of convergence proofs

- Trivial (not the same as easy) to determine convergence either formally or experimentally
- Knowing we have convergence is nice.... but we want to use computers, so we want things to compute fast
- Hence, we often wish to categorize how fast we “converge” towards a solution. This is often termed “rate of convergence”

# The Absolute Error by definition

$$e_n = |x_n - x^*|$$

This will make equations in definitions simpler to read

# Convergence Rate Definitions

**convergence constants...  
smaller is better**



**Linear**  $e_{n+1} \leq c e_n,$

where  $0 < c < 1, n \geq N$

**Super Linear**  $e_{n+1} \leq c_n e_n,$

where  $c_n \rightarrow 0$  for  $n \rightarrow \infty < 1, n \geq N$

**Quadratic**  $e_{n+1} \leq C e_n^2,$

where  $C > 0, n \geq N$

**p-order**  $e_{n+1} \leq C e_n^p,$

where  $C > 0, p > 2, n \geq N$



The “equal” part is the worst behaviour

Holds for  $n$  big enough!

# Linear Convergence Rate Archetype Example

Assume worst-case = worst error reduction => equality part holds

$$e_{n+1} = \frac{1}{2} e_n$$

# Plot “Linear” Archetype Example

scratch\_3.py - PythonIntroduction - [~/PycharmProjects/PythonIntroduction]

string\_formatting.py x scratch\_1.py x scratch\_3.py x scratch\_2.py x

1: Project Task Description

```
1  import matplotlib.pyplot as plt
2
3
4  conv = []
5  e = 1.0
6  c = 0.5
7  for n in range(1,100):
8      e = c * e
9      conv.append(e)
10     plt.plot(conv,'r.',markersize=10)
11     plt.ylabel('value')
12     plt.xlabel('n')
13     plt.show()
```

Run: scratch\_3 scratch\_3 scratch\_3 scratch\_3

4: Run Python Console

Figure 1

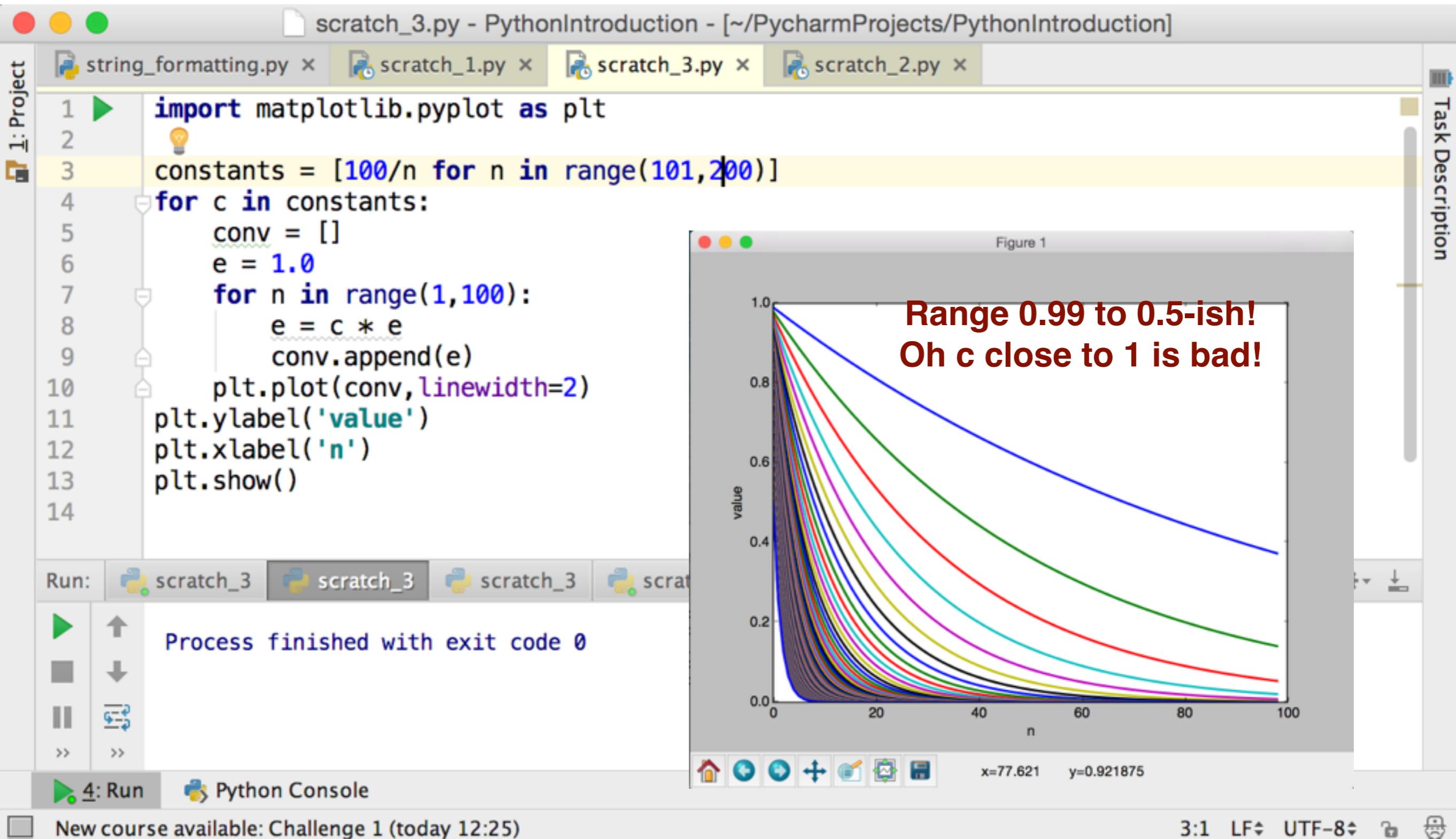
!!!Seems to go pretty fast to zero?

New course available: Challenge 1 (today 12:25)

13:11 LF UTF-8

# Plot “Linear” Archetype Example

## Convergence Constants Play a Big Role



The screenshot shows a PyCharm IDE interface with the following details:

- Project:** 1: Project
- Run:** scratch\_3
- Python Console:** Process finished with exit code 0
- Code Editor:** scratch\_3.py content:

```
1 import matplotlib.pyplot as plt
2
3 constants = [100/n for n in range(101,200)]
4 for c in constants:
5     conv = []
6     e = 1.0
7     for n in range(1,100):
8         e = c * e
9         conv.append(e)
10    plt.plot(conv, linewidth=2)
11 plt.ylabel('value')
12 plt.xlabel('n')
13 plt.show()
```
- Plot:** Figure 1 showing multiple exponential decay curves. The x-axis is labeled 'n' and ranges from 0 to 100. The y-axis is labeled 'value' and ranges from 0.0 to 1.0. A legend in the plot area states: "Range 0.99 to 0.5-ish! Oh c close to 1 is bad!"
- Status Bar:** New course available: Challenge 1 (today 12:25) | 3:1 LF UTF-8

# Linear Convergence Rate Caveats

Given

$$e_{n+1} = \frac{1}{2} e_n$$

for  $n \rightarrow \infty$  we have  $e_n \rightarrow 0$

But if we have an initial error we also observe

$$e_n > 0, \forall n$$

Never going to reach zero error!

# Super-Linear Convergence Rate Archetype Example

Assume worst-case = worst error reduction => equality part holds

$$e_{n+1} = c_n e_n$$

$$c_n = \frac{1}{n}$$

The error will get to zero if run long enough (theoretically)!  
This is different from linear convergence rate

# Super Linear Convergence Rate in Python

scratches\_3.py - PythonIntroduction - [~/PycharmProjects/PythonIntroduction]

string\_formatting.py x scratch\_1.py x scratch\_3.py x scratch\_2.py x

1: Project

```
1 import matplotlib.pyplot as plt
2
3 conv = []
4 const = []
5 e = 1.0
6 for n in range(1,100):
7     c = 1/n
8     e = c * e
9     conv.append(e)
10    const.append(c)
11    plt.plot(conv, linewidth=2)
12    plt.plot(const, linewidth=2)
13    plt.ylabel('value')
14    plt.xlabel('n')
15    plt.show()
```

Run: scratch\_3 scratch\_3 scratch\_3 scratch\_3

Seems faster than linear case....  
but hard to tell apart just by looking

Figure 1

4: Run Python Console

New course available: Challenge 1 (today 12:25)

7:12 LF UTF-8

Task Description

# Quadratic Convergence Rate Archetype Example

Assume worst-case = worst error reduction => equality part holds

$$e_{n+1} = 2 e_n^2$$

# Plot Quadratic Convergence Rate in Python

The screenshot shows a PyCharm IDE interface. The top bar displays the project name "PythonIntroduction" and the file "scratch\_3.py". The code editor contains the following Python script:

```
1 import matplotlib.pyplot as plt
2
3 conv = []
4 e = 0.1
5 for n in range(1,100):
6     e = 2 * e**2
7     conv.append(e)
8 plt.plot(conv, linewidth=2)
9 plt.ylabel('value')
10 plt.xlabel('n')
11 plt.show()
12
```

The run tab at the bottom shows multiple runs of "scratch\_3". To the right, a plot titled "Figure 1" shows a blue curve representing the value of  $e$  versus  $n$ . The x-axis ranges from 0 to 100, and the y-axis ranges from 0.000 to 0.025. The curve starts at approximately (0, 0.02) and drops sharply towards zero, reaching near-zero values very quickly.

**Appears to decrease very fast compared to linear and super-linear  
However, curve has same shape!**

Run: scratch\_3 scratch\_3 scratch\_3 scratch\_3

4: Run Python Console

New course available: Challenge 1 (today 12:25)

# Plot Quadratic Convergence Rate in Python

The screenshot shows a PyCharm interface with the following details:

- Title Bar:** scratch\_3.py - PythonIntroduction - [~/PycharmProjects/PythonIntroduction]
- Toolbars:** Standard red, yellow, green buttons.
- Project Bar:** Shows four files: string\_formatting.py, scratch\_1.py, scratch\_3.py (selected), and scratch\_2.py.
- Code Editor:** Displays the following Python code:

```
1 import matplotlib.pyplot as plt
2
3 conv = []
4 e = 0.6
5 for n in range(1,100):
6     c = 2 * e**2
7     conv.append(e)
8 plt.plot(conv, linewidth=2)
9 plt.ylabel('value')
10 plt.xlabel('n')
11 plt.show()
```

A line containing `e = 0.6` is circled in purple.
- Run Tab:** Shows multiple tabs for scratch\_3.py, with the last one active.
- Run Console:** Displays the error output:

```
File "/Users/kenny/Library/Preferences/PyCharmEdu30/scratches/scratch_3.py", line 6, in <module>
      e = 2 * e**2
OverflowError: (34, 'Result too large')

Process finished with exit code 1
```

A large purple arrow points from the circled line in the editor down to the `OverflowError` message in the console.
- Status Bar:** Shows "New course available: Challenge 1 (today 12:25)" and system status like 7:1, LF, UTF-8, and a lock icon.

# Plot Quadratic Convergence Rate in Python

scratch\_3.py - PythonIntroduction - [~/PycharmProjects/PythonIntroduction]

string\_formatting.py scratch\_1.py scratch\_3.py scratch\_2.py

```
1 import matplotlib.pyplot as plt
2
3 conv = []
4 e = 0.5
5 for n in range(1,100):
6     e = 2 * e**2
7     conv.append(e)
8 plt.plot(conv, linewidth=2)
9 plt.ylabel('value')
10 plt.xlabel('n')
11 plt.show()
12
```

Run: scratch\_3 scratch\_3 scratch\_3 scratch\_3

Process finished with exit code 0

4: Run Python Console

New course available: Challenge 1 (today 12:25)

Figure 1

value

n

Teaser... We will show later that initial e-value must be sufficiently small.

# Can you tell what plots are what rates? Only by looking at the plots?

```
1: Project
1  import matplotlib.pyplot as plt
2
3  conv1 = []
4  conv2 = []
5  conv3 = []
6  e1 = 1.0
7  e2 = 1.0
8  e3 = 0.2
9  for n in range(1,100):
10     e1 = 0.9 * e1
11     e2 = 100/(n+100) * e2
12     e3 = 2 * e3**2
13     conv1.append(e1)
14     conv2.append(e2)
15     conv3.append(e3)
16     plt.plot(conv1, linewidth=2, label='Linear')
17     plt.plot(conv2, linewidth=2, label='Super')
18     plt.plot(conv3, linewidth=2, label='Quadratic')
19     plt.legend()
20     plt.ylabel('value')
21     plt.xlabel('n')
22     plt.show()
```

Figure 1

value

n

Linear  
Super  
Quadratic

x=65.9274 y=0.84375

Run: scratch\_3 scratch\_3 scratch\_3 scratch\_3

4: Run Python Console

New course available: Challenge 1 (today 12:25)

11:26 LF UTF-8

We wish to be able to classify  
the convergence rate “easily”

- It will allow us to verify if our implementations are working as theory tell us we should expect them to work (Experimentally verification of convergence theorems)
- Notice the “proof” of a certain rate is only half of the story... The other is whether one code delivers on this too!!!

# Linear Convergence Rate Again

$$e_1 = c e_0$$

$$e_2 = c^2 e_0$$

$$e_3 = c^3 e_0$$

⋮  
⋮

$$e_n = c^n e_0$$

Taking the logarithm

$$\log e_n = \log (c^n e_0)$$

$$\log e_n = \log c^n + \log e_0$$

$$\underbrace{\log e_n}_y = \underbrace{(\log c)}_a \underbrace{n}_x + \underbrace{\log e_0}_b$$

We have a straight-line in a log plot

# Quadratic Convergence Rate Again

$$e_1 = C e_0^2$$

$$e_2 = C e_1^2 = C^3 e_0^4$$

⋮

$$e_n = C e_{n-1}^2 = C^{2^n - 1} e_0^{2^n}$$

Taking logarithm

$$\log e_n = \log \left( C^{2^n - 1} e_0^{2^n} \right)$$

$$\log e_n = \log \left( C^{2^n - 1} \right) + \log \left( e_0^{2^n} \right)$$

$$\log e_n = (2^n - 1) \log C + 2^n \log e_0$$

$$\log e_n = 2^n \log (C e_0) - \log C$$



**Important!**  
**C must be**  
**positive**

# Quadratic Convergence Rate Again

So

$$\log e_n = \log \left( C^{2^n - 1} e_0^{2^n} \right)$$

$$\log e_n = \log \left( C^{2^n - 1} \right) + \log \left( e_0^{2^n} \right)$$

$$\log e_n = (2^n - 1) \log C + 2^n \log e_0$$

$$\log e_n = 2^n \log (Ce_0) - \log C$$

We must have

$$e_n \rightarrow 0 \Rightarrow \log(e_n) \rightarrow -\infty$$

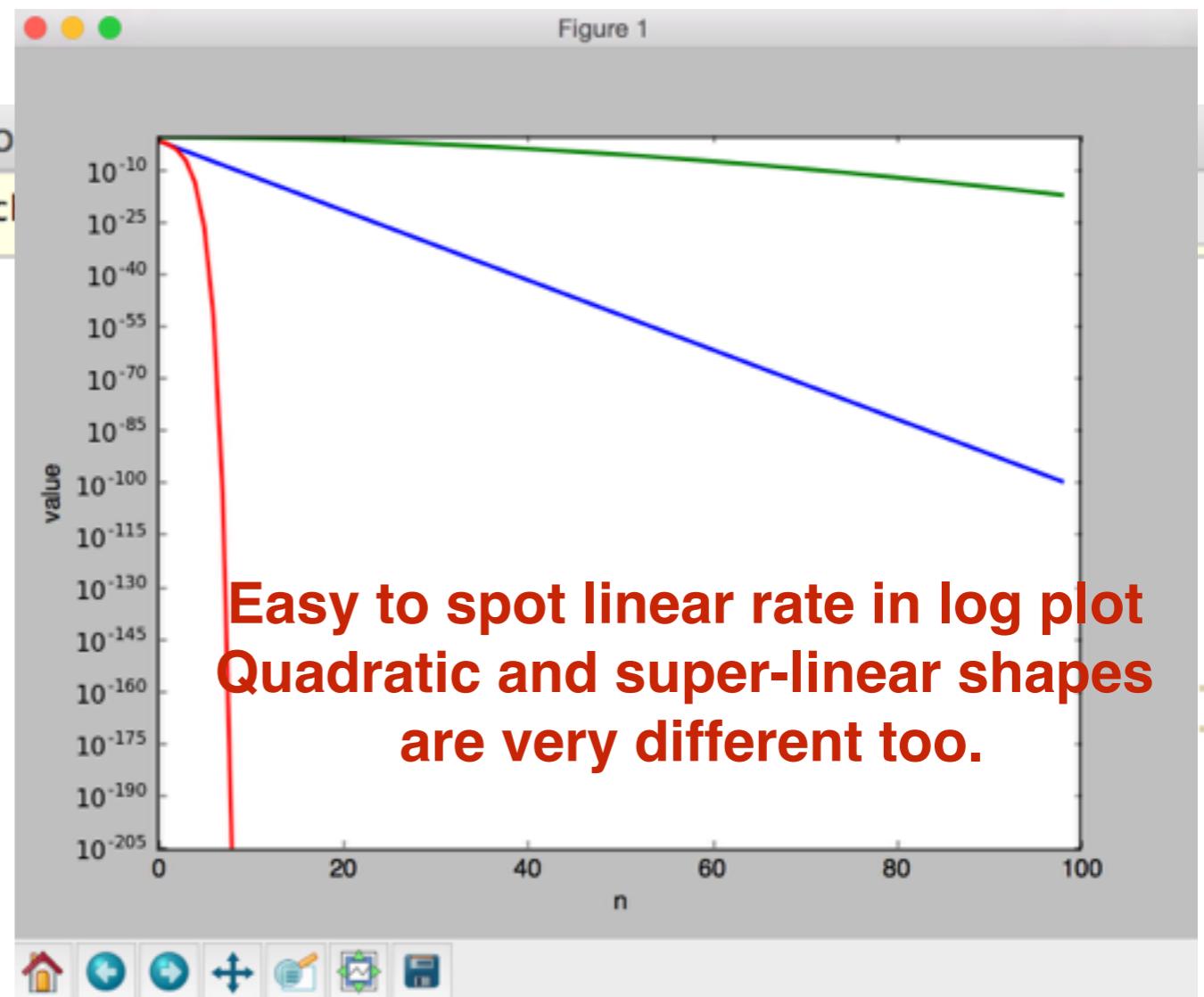
Only possible if

$$\log (Ce_0) < 0 \quad \text{or} \quad e_0 < \frac{1}{C}$$

**Important!**  
**Initial error has**  
**to be small**  
**enough!**

# Log-Scale Shows Linear Convergence Rate as a straight line

```
scratch_3.py - PythonIntroduction
string_formatting.py x scratch_1.py x scratch_3.py
1: Project
1 import matplotlib.pyplot as plt
2
3 conv1 = []
4 conv2 = []
5 conv3 = []
6 e1 = 1.0
7 e2 = 1.0
8 e3 = 0.2
9 for n in range(1,100):
10     e1 = 0.1 * e1
11     e2 = 100/(n+100) * e2
12     e3 = 2 * e3**2
13     conv1.append(e1)
14     conv2.append(e2)
15     conv3.append(e3)
16 plt.semilogy(conv1, linewidth=2, label='Linear')
17 plt.semilogy(conv2, linewidth=2, label='Super')
18 plt.semilogy(conv3, linewidth=2, label='Quadratic')
19 plt.ylabel('value')
20 plt.xlabel('n')
21 plt.show()
22
```



Can we get straight lines for quadratic convergence rate?

We have

$$\log e_n = 2^n \log(Ce_0) - \log C$$

Observe

$$e_n \rightarrow 0 \Rightarrow \log e_n \rightarrow -\infty$$

Means  $\log(\log e_n)$  is undefined

If we flip sign then

$$\log(-\log e_n) = \log(-2^n \log Ce_n + \log(\log C))$$

This will never be affine function in  $n$

Can we get straight lines for quadratic convergence rate?

Let us make a substitution

$$\log e_n = \underbrace{2^n}_{\equiv k} \log C e_n - \log C$$

If we plot log error as a function of  $k$  then we have a affine function

# Quadratic convergence rate as a straight line in Python

The screenshot shows a Python development environment with the following components:

- Project Bar:** Shows files scratch\_1.py, scratch\_4.py, and scratch\_8.py.
- Code Editor:** Displays the following Python code:

```
1: Project
2: scratch_8.py - Python
3:
4: C = 2.0
5: e = 0.4
6: y = []
7: k = []
8: for n in range(1,12):
9:     e = C* e**2
10:    y.append(e)
11:    k.append(2**n)
12: plt.semilogy(k,y)
13: plt.show()
```
- Run Bar:** Shows the output of the run command: [2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048] and [0.3200000000000006, 0.2048000000000001, 0.08388608000000007, 0.014073748835532824, 0.000396140812571323].
- Figure Window:** Titled "Figure 1", it contains a log-linear plot of error  $e$  versus iteration  $k$ . The y-axis is logarithmic, ranging from  $10^{-199}$  to  $10^{-4}$ . The x-axis ranges from 0 to 2500. A single blue line shows a perfect linear relationship, indicating quadratic convergence.
- Task Description:** A red annotation text "So it is possible... but a bit impractical" is overlaid on the plot area.

# Big O definition for Sequences

We write

$$x_n = \mathcal{O}(y_n) \quad n \rightarrow \infty$$

$$x_n = \mathcal{O}(y_n) \quad n \rightarrow 0$$

$$x_n = \mathcal{O}(y_n) \quad n \rightarrow n^*$$

This means there exist  $C > 0$  and  $N > 0$  such that

$$|x_n| \leq C|y_n| \quad n \geq N$$

$$|x_n| \leq C|y_n| \quad n \leq N$$

$$|x_n| \leq C|y_n| \quad |n^* - n| \leq N$$

# Big O Example

$$\frac{5 \cos(n)}{n^2} = \mathcal{O}\left(\frac{1}{n}\right) \quad n \rightarrow \infty$$

# Big O Example in Python

scratch\_3.py - PythonIntroduction - [~/PycharmProjects/PythonIntroduction]

1: Project

string\_formatting.py x scratch\_1.py x scratch\_3.py x scratch\_2.py x

```
1 > import matplotlib.pyplot as plt
2   from math import cos
3
4   f = []
5   gp = []
6   gm = []
7   for n in range(1,100):
8       f.append( 5*cos(n) / n**2 )
9       gp.append( 1 / n )
10      gm.append( -1 / n )
11      plt.plot(f, linewidth=2)
12      plt.plot(gp, linewidth=2)
13      plt.plot(gm, linewidth=2)
14      plt.ylabel('value')
15      plt.xlabel('n')
16      plt.show()
17
```

Run: scratch\_3 scratch\_3 scratch\_3 scratch

4: Run Python Console

New course available: Challenge 1 (today 12:25)

Figure 1

It is a sandwich!

8:17 LF UTF-8

# Example Using Big “O” to tell how cost scales

Given  $\vec{x}, \vec{y} \in \mathbf{R}^n$  what is the computational cost of

$$\text{cost}(\vec{x} \cdot \vec{y}) = \mathcal{O}(?)$$

Answer

$$\text{cost}(\vec{x} \cdot \vec{y}) = \mathcal{O}(n)$$

Given  $A \in \mathbf{R}^{m \times n}$  what is the computational cost of

$$\text{cost}(A \vec{x}) = \mathcal{O}(?)$$

The  $n$  is called the problem size and big O tells how computations scale when  $n$  grows

# Limit Cases Often Encountered when Analysing Errors and Costs

	Functions	Sequences
Infinity	$\lim_{x \rightarrow \infty} f(x) = c$	$\lim_{n \rightarrow \infty} x_n = c$
Value	$\lim_{x \rightarrow x^*} f(x) = c$	$\lim_{n \rightarrow 0} x_n = c$

# What is a Limit?

We write

$$\lim_{x \rightarrow x^*} f(x) = c$$

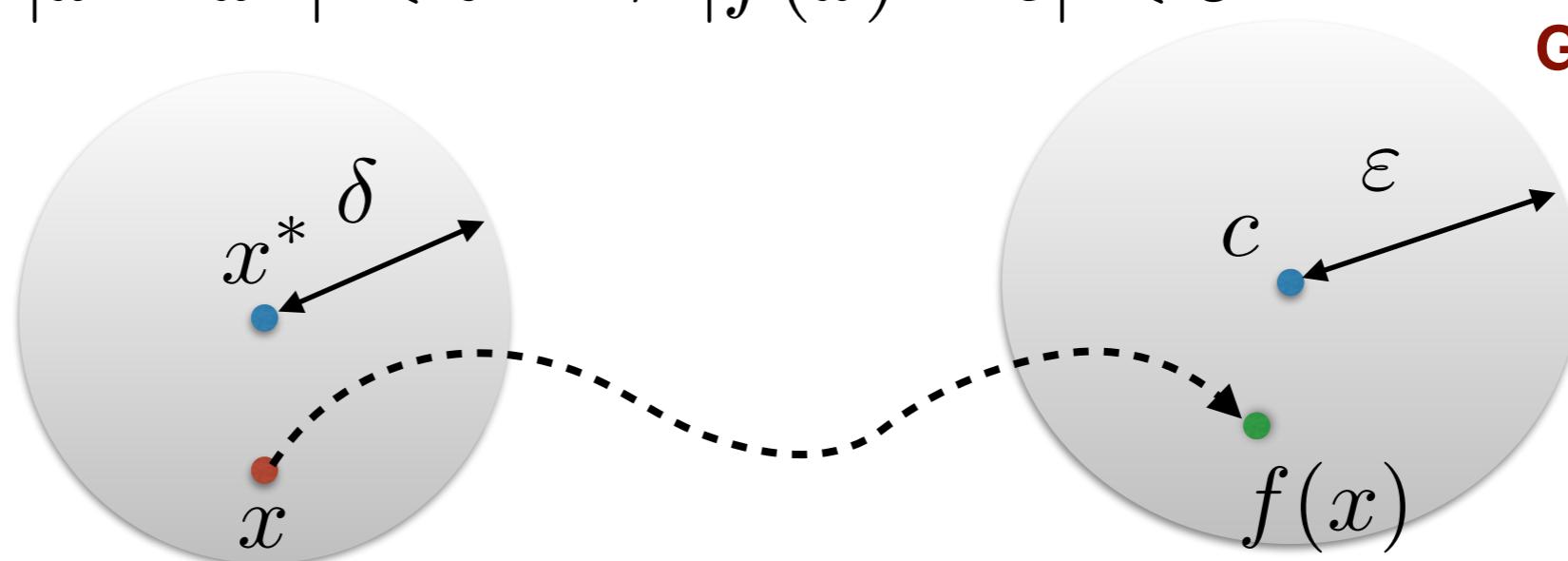
If and only if

$$\forall \varepsilon > 0, \exists \delta > 0$$

such that

$$|x - x^*| < \delta \Rightarrow |f(x) - c| < \varepsilon$$

**Got to prove  
 $\delta$  exists!**



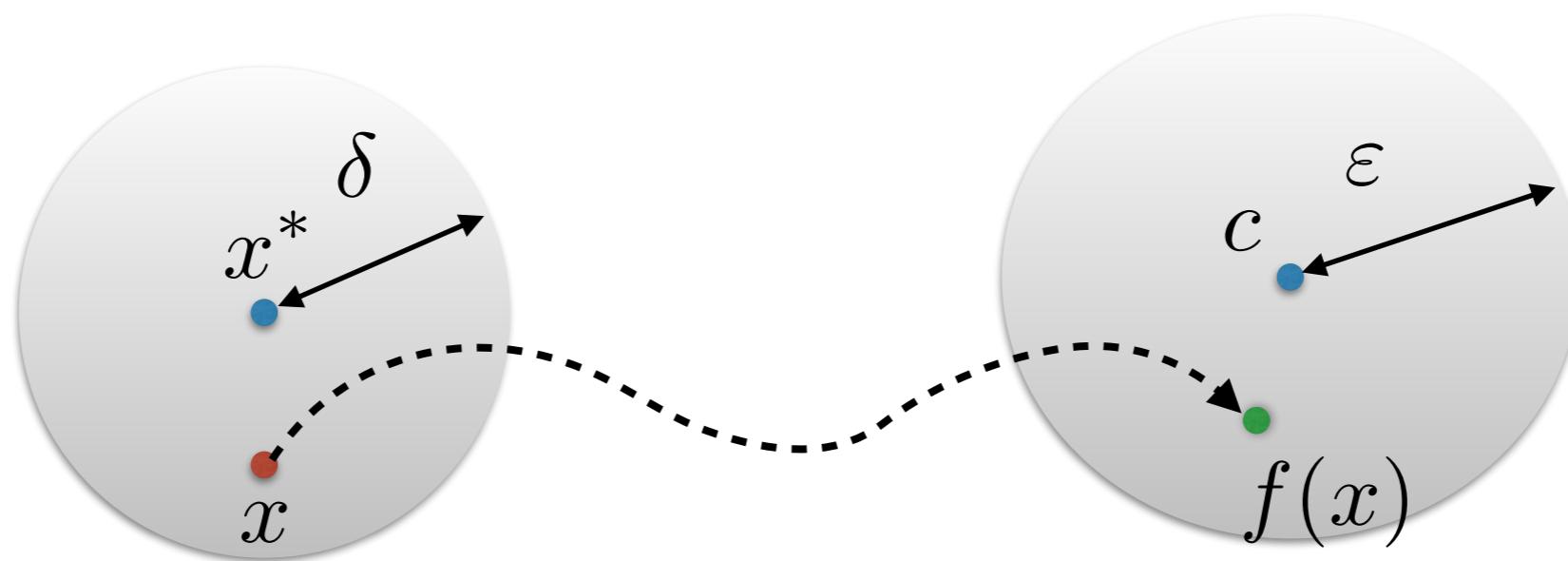
# Geometric Interpretation of Limit

$$\lim_{x \rightarrow x^*} f(x) = c$$

Means

$$x \in \mathcal{B}_\delta(x^*) \Rightarrow f(x) \in \mathcal{B}_\varepsilon(c)$$

Got to prove  
 $\delta$  exists!



Given any  $\varepsilon$

# Limit Example

Prove for some fixed, positive integer  $n$ ,

$$\lim_{x \rightarrow 0} x^n = 0$$

Observe  $x^* = c = 0$       Pick any  $\varepsilon > 0$

$$|x^n - c| = |x^n| = |x|^n < \varepsilon$$

$$|x - x^*| = |x - 0| = |x| < \sqrt[n]{\varepsilon}$$

$$\delta = \sqrt[n]{\varepsilon}$$

We proved that  $\delta$  exists

# Little o definition for sequences

We write

$$x_n = \mathbf{o}(y_n) \quad n \rightarrow \infty$$

$$x_n = \mathbf{o}(y_n) \quad n \rightarrow 0$$

$$x_n = \mathbf{o}(y_n) \quad n \rightarrow n^*$$

This means

$$\lim_{n \rightarrow \infty} \left( \frac{x_n}{y_n} \right) = 0$$

$$\lim_{n \rightarrow 0} \left( \frac{x_n}{y_n} \right) = 0$$

$$\lim_{n \rightarrow n^*} \left( \frac{x_n}{y_n} \right) = 0$$

Interpretation x's goes more rapid to zero than y's