In [5]:
```
from __future__ import print_function
%matplotlib inline
import ganymede
ganymede.configure('uav.beaver.works')
import matplotlib.pyplot as plt
import numpy as np
import sympy as sym
from IPython.display import YouTubeVideo, HTML
sym.init_printing(use_latex = "mathjax")
```

```
/home/saanvi/.local/lib/python3.10/site-packages/matplotlib/projections/__init__.py:63: UserWarning: Unable to import Axes3D. This may be due to multiple versions of Matplotlib being installed (e.g. as a system package and as a pip package). As a result, the 3D projection is not availabl
e.
  warnings.warn("Unable to import Axes3D. This may be due to multiple versions of "
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[5], line 3
      1 from __future__ import print_function
      2 get_ipython().run_line_magic('matplotlib', 'inline')
----> 3 import ganymede
      4 ganymede.configure('uav.beaver.works')
      5 import matplotlib.pyplot as plt

ModuleNotFoundError: No module named 'ganymede'
```

Enter your name below and run the cell:

Individual cells can be run with `Ctrl` + `Enter`

In [8]:
```
# ganymede.name('Saanvi Chugh')
# def check(p):
#     # ganymede.update(p,True)
# check(0)
```

In [3]:
```
YouTubeVideo('9vKqVkMQHKk', width=560, height=315) # Video by http://www.3blue1brown.com/
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[3], line 1
----> 1 YouTubeVideo('9vKqVkMQHKk', width=560, height=315) # Video by http://www.3blue1brown.com/

NameError: name 'YouTubeVideo' is not defined
```

In [ ]:
```
YouTubeVideo('bRZmfc1YFsQ', width=560, height=315) #Note: All Khan Academy content is available for free at khanacademy.org
```

Power Rule

The derivative of x^n is nx^{n-1}

Read more

Other derivative rules

In [4]:
```
# Creating algebraic symbols
x = sym.symbols('x')
x
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[4], line 2
      1 # Creating algebraic symbols
----> 2 x = sym.symbols('x')
      3 x

NameError: name 'sym' is not defined
```

In [5]:
```
x = sym.symbols('x')
expr = x ** 2
expr
```

Out[5]:

$$x^2$$

In [6]:
```
sym.Derivative(expr) # does not actually compute the derivative
```

Out[6]:

$$\frac{d}{dx}x^2$$

In [7]:
```
sym.Derivative(expr).doit()
```

Out[7]:

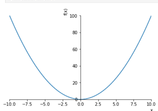
$$2x$$

In [8]:
```
sym.diff(expr) #equivalent to doit()
```

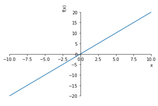
Out[8]:

$$2x$$

In [9]:
```
sym.plot(expr);
```



In [10]:
```
sym.plot(sym.diff(expr));
```



In [ ]:
```
x = sym.symbols('x')
expr = -x ** 3 + 2

sym.plot(expr, xlim=(-2, 2), ylim=(-10, 10));
```

In [12]:
```
sym.Derivative(expr)
```

Out[12]:

$$\frac{d}{dx}\left(-x^3+2\right)$$

In [13]:
```
sym.Derivative(expr).doit()
```

Out[13]:

$$-3x^2$$

In [ ]:
```
sym.plot(sym.diff(expr));
```

Now, let's generate a fake one-dimensional signal:

In [9]:
```
ys = np.array([0, 1, 0, 1, 2, 0, 2, 3, 2, 1, 2, 102, 100,  95, 100,  98,  99, 104, 110, 103, 100,  96, 102, 101])

fig,ax = plt.subplots()
ax.plot([i for i in range(len(ys))], ys);
# check(1)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[9], line 1
----> 1 ys = np.array([0, 1, 0, 1, 2, 0, 2, 3, 2, 1, 2, 102, 100,  95, 100,  98,  99, 104, 110, 103, 100,  96, 102, 101])
      3 fig,ax = plt.subplots()
      4 ax.plot([i for i in range(len(ys))], ys);

NameError: name 'np' is not defined
```

Next, let's look at small chunks of our fake signal:

In [ ]:
```
chunks = np.split(ys, len(ys)//2)
print(chunks)
check(2)
```

Question: Which one of these chunks would you say is the most "interesting"? Between 2 and 102 is the most interesting because this is where it spikes up.

Question If we always divide up the signal as we did above, will we always find something "interesting"? Not necessarily, it could just be a straight line.

Convolutions

Derivatives and convolutions are one technique to help us tackle the above problem.

First, you'll need to generate windows into the signal. Write a function that can generate windows with a user-supplied windowsize, and print them out.

An example signal with 3 window sizes is shown below. Your output does not need to replicate the formatting shown, but they should produce the same windows. E.g., given an input signal of `[10,20,30]` and a `windowsize=2`, your function should return `[[10,20], [20,30]]`.

A windowsize of 1:

```
signal:
        0   1   0   2   1   0   1 101 100  98 102 101
0:      0
1:  ____    1
2:  ________    0
3:  ____________    2
4:  ________________    1
5:  ____________________    0
6:  ________________________    1
7:  ____________________________    101
8:  ________________________________    100
9:  ____________________________________    98
10: ________________________________________    102
11: ____________________________________________    101

          ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

i:    0 |  i + windowsize:    1   |  window:  [   0]
i:    1 |  i + windowsize:    2   |  window:  [   1]
i:    2 |  i + windowsize:    3   |  window:  [   0]
i:    3 |  i + windowsize:    4   |  window:  [   2]
i:    4 |  i + windowsize:    5   |  window:  [   1]
i:    5 |  i + windowsize:    6   |  window:  [   0]
i:    6 |  i + windowsize:    7   |  window:  [   1]
i:    7 |  i + windowsize:    8   |  window:  [ 101]
i:    8 |  i + windowsize:    9   |  window:  [ 100]
i:    9 |  i + windowsize:   10   |  window:  [  98]
i:   10 |  i + windowsize:   11   |  window:  [ 102]
i:   11 |  i + windowsize:   12   |  window:  [ 101]
```

A windowsize of 2:

```
signal:
        0   1   0   2   1   0   1 101 100  98 102 101
0:      0   1
1:  ____    1   0
2:  ________    0   2
3:  ____________    2   1
4:  ________________    1   0
5:  ____________________    0   1
6:  ________________________    1 101
7:  ____________________________    101 100
8:  ________________________________    100  98
9:  ____________________________________    98 102
10: ________________________________________    102 101

          ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

i:    0 |  i + windowsize:    2   |  window:  [   0,   1]
i:    1 |  i + windowsize:    3   |  window:  [   1,   0]
i:    2 |  i + windowsize:    4   |  window:  [   0,   2]
i:    3 |  i + windowsize:    5   |  window:  [   2,   1]
i:    4 |  i + windowsize:    6   |  window:  [   1,   0]
i:    5 |  i + windowsize:    7   |  window:  [   0,   1]
i:    6 |  i + windowsize:    8   |  window:  [   1, 101]
i:    7 |  i + windowsize:    9   |  window:  [ 101, 100]
i:    8 |  i + windowsize:   10   |  window:  [ 100,  98]
i:    9 |  i + windowsize:   11   |  window:  [  98, 102]
i:   10 |  i + windowsize:   12   |  window:  [ 102, 101]
```

A windowsize of 3

```
signal:
        0   1   0   2   1   0   1 101 100  98 102 101
0:      0   1   0
1:  ____    1   0   2
2:  ________    0   2   1
3:  ____________    2   1   0
4:  ________________    1   0   1
5:  ____________________    0   1 101
6:  ________________________    1 101 100
7:  ____________________________    101 100  98
8:  ________________________________    100  98 102
9:  ____________________________________    98 102 101

          ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

i:    0 |  i + windowsize:    3   |  window:  [   0,   1,   0]
i:    1 |  i + windowsize:    4   |  window:  [   1,   0,   2]
i:    2 |  i + windowsize:    5   |  window:  [   0,   2,   1]
i:    3 |  i + windowsize:    6   |  window:  [   2,   1,   0]
i:    4 |  i + windowsize:    7   |  window:  [   1,   0,   1]
i:    5 |  i + windowsize:    8   |  window:  [   0,   1, 101]
i:    6 |  i + windowsize:    9   |  window:  [   1, 101, 100]
i:    7 |  i + windowsize:   10   |  window:  [ 101, 100,  98]
i:    8 |  i + windowsize:   11   |  window:  [ 100,  98, 102]
i:    9 |  i + windowsize:   12   |  window:  [  98, 102, 101]
```

The below resources may be helpful:

List Comprehensions

https://www.pythonlikeyoumeanit.com/Module2_EssentialsOfPython/Generators_and_Comprehensions.html#List-&-Tuple-Comprehensions

Numpy indexing with slices

http://www.pythonlikeyoumeanit.com/Module3_IntroducingNumpy/AccessingDataAlongMultipleDimensions.html#Slice-Indexing

Formatting numbers in python

https://pyformat.info/#number

input: `'{:4d}'.format(42)`

output: `_ _ 4 2`

input: `'{:06.2f}'.format(3.1415926535897931)`

output: `003.14`

String concatenation

```
>>> print('a' + 'b' + 'c')
abc
>>> print(''.join(['a', 'b', 'c']))
abc
>>> print(''.join(['a', 'b', 'c']))
a,b,c
```

In [ ]:
```
def make_windows(sequence, windowsize):
    window = []
    for i in range(len(sequence) - windowsize + 1):
        new_list = sequence[i:i + windowsize]
        window.append(new_list)
    print(window)

make_windows([10,20,30], 2)
```

In [ ]:
```
series = [0, 1, 0, 2, 1, 0, 1, 101, 100, 98, 102, 101]


make_windows(sequence=series, windowsize=1)
make_windows(sequence=series, windowsize=2)
make_windows(sequence=series, windowsize=3)

check(3)
```

When you are done:

Generate some example outputs in this notebook.

1. Double-check that you filled in your name at the top of the notebook!

2. Click `File -> Export Notebook As -> PDF`
3. Email the PDF to `YOURTEAMNAME@beaver.works`