In [1]:
```
import numpy as np
array = np.array([1, 2, 3, 4, 5])
array
```

Out[1]:
```
array([1, 2, 3, 4, 5])
```

In [ ]:
```
np.log([1, 2, 3, 4, 5])
```

In [2]:
```
import random
rand_list = [random.random() * 10 for i in range (10)]
rand_list
```

Out[2]:
```
[9.972602428554506,
 7.476930522338399,
 5.662720979969801,
 6.360988255439208,
 1.6963974433624684,
 3.843993123441738,
 7.931911088436992,
 3.970103310912548,
 1.6896765151719018,
 7.516102700797975]
```

In [ ]:
```

```

In [3]:
```
subset = [val for val in rand_list if val > 7]
subset
```

Out[3]:
```
[9.972602428554506, 7.476930522338399, 7.931911088436992, 7.516102700797975]
```

In [4]:
```
import numpy as np
rand_array = np.random.randn(7) * 10
print (rand_array)
print (rand_array > 0)
```

```
[ 2.89853047  3.82740831 -0.75430288 -3.9873432  -4.62567324 14.89704623
  2.68142211]
[ True  True False False False  True  True]
```

In [5]:
```
import numpy as np
rand_array = np.random.randn(7) * 10
print (rand_array)
print (rand_array > 0)
rand_array[rand_array > 0]
```

```
[  9.96790587  -8.09372932  -2.19576085 -18.29458287  -4.19389997
  -5.42160463 -13.4710504 ]
[ True False False False False False False]
```

Out[5]:
```
array([9.96790587])
```

In [6]:
```
data_dict = {"0 to 9" : np.arange(10),
 "ones" : np.ones(10),
 "zeros" : np.zeros(10)}
print(data_dict)
```

```
{'0 to 9': array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]), 'ones': array([1., 1., 1.,
1., 1., 1., 1., 1., 1., 1.]), 'zeros': array([0., 0., 0., 0., 0., 0., 0., 0.,
0., 0.])}
```

In [7]:
```
for key in data_dict:
 print(key)
```

```
0 to 9
ones
zeros
```

In [ ]:
```
```

In [8]:
```
for key in data_dict:
 print(key)
 print(data_dict[key])
data_dict.items()
```

```
0 to 9
[0 1 2 3 4 5 6 7 8 9]
ones
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
zeros
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

Out[8]:
```
dict_items([('0 to 9', array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])), ('ones', array
([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])), ('zeros', array([0., 0., 0., 0.,
0., 0., 0., 0., 0., 0.]))])
```

In [ ]:
```
```

In [9]:
```
for key in data_dict:
 val = data_dict[key]
 print(key)
 print(val)
# print(data_dict[key])
data_dict.items()
```

```
0 to 9
[0 1 2 3 4 5 6 7 8 9]
ones
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
zeros
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

Out[9]:
```
dict_items([('0 to 9', array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])), ('ones', array
([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])), ('zeros', array([0., 0., 0., 0.,
0., 0., 0., 0., 0., 0.]))])
```

In [ ]:
```

```

In [10]:
```
for key, val in data_dict.items():
 print(key)
 print(val)
```

```
0 to 9
[0 1 2 3 4 5 6 7 8 9]
ones
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
zeros
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

In [ ]:
```

```

In [11]:
```
for key, val in data_dict.items():
 print("Values from index 5", key)
 print(val[5:])
```

```
Values from index 5 0 to 9
[5 6 7 8 9]
Values from index 5 ones
[1. 1. 1. 1. 1.]
Values from index 5 zeros
[0. 0. 0. 0. 0.]
```

In [ ]:
```

```

In [12]:
```
import pandas as pd
data = pd.Series(np.arange(2,12))
print(data)
```

```
0     2
1     3
2     4
3     5
4     6
5     7
6     8
7     9
8    10
9    11
dtype: int32
```

In [ ]:

In [14]:
```
data_df = pd.DataFrame(data_dict)
data_df
```

Out[14]:

	0 to 9	ones	zeros
0	0	1.0	0.0
1	1	1.0	0.0
2	2	1.0	0.0
3	3	1.0	0.0
4	4	1.0	0.0
5	5	1.0	0.0
6	6	1.0	0.0
7	7	1.0	0.0
8	8	1.0	0.0
9	9	1.0	0.0

In [ ]:

In [ ]:

In [15]:
```
data_df["0 to 9"]
```

Out[15]:
```
0    0
1    1
2    2
3    3
4    4
5    5
6    6
7    7
8    8
9    9
Name: 0 to 9, dtype: int32
```

In [ ]:
```

```

In [16]:
```
#df.loc[start:finish - 1] calls according to row name
data_df.loc[:5]
```

Out[16]:

	0 to 9	ones	zeros
0	0	1.0	0.0
1	1	1.0	0.0
2	2	1.0	0.0
3	3	1.0	0.0
4	4	1.0	0.0
5	5	1.0	0.0

In [ ]:
```

```

In [17]:
```
#df.loc[start:finish - 1] calls according to row name
data_df.loc[:5]
```

Out[17]:

	0 to 9	ones	zeros
0	0	1.0	0.0
1	1	1.0	0.0
2	2	1.0	0.0
3	3	1.0	0.0
4	4	1.0	0.0
5	5	1.0	0.0

In [ ]:
```

```

In [22]:
```
macro_dict = {"GDP" : {},
 "Money" : {},
 "Real GDP" : {},
 "Price Level" : {}}
for key in macro_dict:
    for i in range(1990, 2010):
        macro_dict[key][i] = np.random.random() * 10000

print(macro_dict)
```

```
{'GDP': {1990: 6852.146663878849, 1991: 8910.248857741317, 1992: 5070.317477707
74, 1993: 2040.9402721313863, 1994: 6782.45243067946, 1995: 6099.269119929569,
1996: 1208.9417876369935, 1997: 2122.2888641217587, 1998: 6647.020909042928, 19
99: 4165.493106468629, 2000: 3190.2196208665646, 2001: 9343.647429047012, 2002:
4813.430709787839, 2003: 5684.6953724452, 2004: 2539.473666766634, 2005: 4512.7
98062049088, 2006: 3169.884401150863, 2007: 6802.157772886638, 2008: 8897.96351
7196496, 2009: 4962.447979026464}, 'Money': {1990: 2738.5912051946084, 1991: 28
53.0115005441203, 1992: 2609.317444651117, 1993: 6819.699511494199, 1994: 1364.
3766245197053, 1995: 8403.946529825664, 1996: 6334.065847304254, 1997: 1687.066
1041063906, 1998: 2150.7482757325633, 1999: 4388.539561189525, 2000: 7996.24727
2770763, 2001: 581.3578925109974, 2002: 4814.917413305774, 2003: 8785.949532040
664, 2004: 8627.481069321566, 2005: 8469.561724257512, 2006: 8706.915644586677,
2007: 2889.6285342052074, 2008: 5765.731528850918, 2009: 2209.5756091396192},
'Real GDP': {1990: 9768.169504472622, 1991: 5487.335071845204, 1992: 7780.01007
7552613, 1993: 9709.574074111863, 1994: 8459.612459026614, 1995: 6311.389968411
903, 1996: 1477.8021069390645, 1997: 7727.2096434124105, 1998: 5970.65246342435
9, 1999: 4457.210691376601, 2000: 2809.5321862278633, 2001: 2749.2804956744, 20
02: 8535.138662054222, 2003: 7033.7791425867745, 2004: 6297.448472138756, 2005:
5425.381294882602, 2006: 7375.258125789222, 2007: 8485.30695401491, 2008: 7330.
481930187528, 2009: 2129.090384811797}, 'Price Level': {1990: 2271.806049025473
4, 1991: 82.62352712498978, 1992: 2022.5971360676122, 1993: 3933.029797228649,
1994: 477.4400262305134, 1995: 6987.789929769195, 1996: 4452.037561727889, 199
7: 413.7172442278692, 1998: 9153.0549864842, 1999: 6720.744495462311, 2000: 128
0.1292587760272, 2001: 3937.3726631217587, 2002: 8786.779709426954, 2003: 3497.
11369794951, 2004: 5652.880899614199, 2005: 82.57944080277025, 2006: 5376.30322
11652635, 2007: 3097.0146848813506, 2008: 9456.385450640586, 2009: 5503.1694870
58234}}
```

In [ ]:

In [ ]:

In [23]:
```
data_df = pd.DataFrame(macro_dict)
data_df
```

Out[23]:

	GDP	Money	Real GDP	Price Level
1990	6852.146664	2738.591205	9768.169504	2271.806049
1991	8910.248858	2853.011501	5487.335072	82.623527
1992	5070.317478	2609.317445	7780.010078	2022.597136
1993	2040.940272	6819.699511	9709.574074	3933.029797
1994	6782.452431	1364.376625	8459.612459	477.440026
1995	6099.269120	8403.946530	6311.389968	6987.789930
1996	1208.941788	6334.065847	1477.802107	4452.037562
1997	2122.288864	1687.066104	7727.209643	413.717244
1998	6647.020909	2150.748276	5970.652463	9153.054986
1999	4165.493106	4388.539561	4457.210691	6720.744495
2000	3190.219621	7996.247273	2809.532186	1280.129259
2001	9343.647429	581.357893	2749.280496	3937.372663
2002	4813.430710	4814.917413	8535.138662	8786.779709
2003	5684.695372	8785.949532	7033.779143	3497.113698
2004	2539.473667	8627.481069	6297.448472	5652.880900
2005	4512.798062	8469.561724	5425.381295	82.579441
2006	3169.884401	8706.915645	7375.258126	5376.303221
2007	6802.157773	2889.628534	8485.306954	3097.014685
2008	8897.963517	5765.731529	7330.481930	9456.385451
2009	4962.447979	2209.575609	2129.090385	5503.169487

In [ ]:

In [ ]:

In [24]:
```
macro_df = pd.DataFrame(macro_dict)
macro_df
```

Out[24]:

	GDP	Money	Real GDP	Price Level
1990	6852.146664	2738.591205	9768.169504	2271.806049
1991	8910.248858	2853.011501	5487.335072	82.623527
1992	5070.317478	2609.317445	7780.010078	2022.597136
1993	2040.940272	6819.699511	9709.574074	3933.029797
1994	6782.452431	1364.376625	8459.612459	477.440026
1995	6099.269120	8403.946530	6311.389968	6987.789930
1996	1208.941788	6334.065847	1477.802107	4452.037562
1997	2122.288864	1687.066104	7727.209643	413.717244
1998	6647.020909	2150.748276	5970.652463	9153.054986
1999	4165.493106	4388.539561	4457.210691	6720.744495
2000	3190.219621	7996.247273	2809.532186	1280.129259
2001	9343.647429	581.357893	2749.280496	3937.372663
2002	4813.430710	4814.917413	8535.138662	8786.779709
2003	5684.695372	8785.949532	7033.779143	3497.113698
2004	2539.473667	8627.481069	6297.448472	5652.880900
2005	4512.798062	8469.561724	5425.381295	82.579441
2006	3169.884401	8706.915645	7375.258126	5376.303221
2007	6802.157773	2889.628534	8485.306954	3097.014685
2008	8897.963517	5765.731529	7330.481930	9456.385451
2009	4962.447979	2209.575609	2129.090385	5503.169487

In [ ]:

In [ ]:

In [25]:
```
macro_df["Velocity"] = macro_df["GDP"] / macro_df["Money"]
macro_df["Real GDP"] = macro_df["GDP"] / macro_df["Price Level"]
print(macro_df)
```

```
                GDP        Money    Real GDP  Price Level   Velocity
1990   6852.146664  2738.591205    3.016167  2271.806049   2.502070
1991   8910.248858  2853.011501  107.841545    82.623527   3.123103
1992   5070.317478  2609.317445    2.506835  2022.597136   1.943159
1993   2040.940272  6819.699511    0.518923  3933.029797   0.299271
1994   6782.452431  1364.376625   14.205873   477.440026   4.971100
1995   6099.269120  8403.946530    0.872847  6987.789930   0.725762
1996   1208.941788  6334.065847    0.271548  4452.037562   0.190863
1997   2122.288864  1687.066104    5.129805   413.717244   1.257976
1998   6647.020909  2150.748276    0.726208  9153.054986   3.090562
1999   4165.493106  4388.539561    0.619796  6720.744495   0.949175
2000   3190.219621  7996.247273    2.492107  1280.129259   0.398965
2001   9343.647429   581.357893    2.373067  3937.372663  16.072109
2002   4813.430710  4814.917413    0.547804  8786.779709   0.999691
2003   5684.695372  8785.949532    1.625539  3497.113698   0.647021
2004   2539.473667  8627.481069    0.449235  5652.880900   0.294347
2005   4512.798062  8469.561724   54.647961    82.579441   0.532825
2006   3169.884401  8706.915645    0.589603  5376.303221   0.364065
2007   6802.157773  2889.628534    2.196360  3097.014685   2.353990
2008   8897.963517  5765.731529    0.940948  9456.385451   1.543250
2009   4962.447979  2209.575609    0.901744  5503.169487   2.245883
```

In [ ]:

In [ ]:

In [26]:
```
macro_df.loc[2002:2009]
```

Out[26]:

	GDP	Money	Real GDP	Price Level	Velocity
2002	4813.430710	4814.917413	0.547804	8786.779709	0.999691
2003	5684.695372	8785.949532	1.625539	3497.113698	0.647021
2004	2539.473667	8627.481069	0.449235	5652.880900	0.294347
2005	4512.798062	8469.561724	54.647961	82.579441	0.532825
2006	3169.884401	8706.915645	0.589603	5376.303221	0.364065
2007	6802.157773	2889.628534	2.196360	3097.014685	2.353990
2008	8897.963517	5765.731529	0.940948	9456.385451	1.543250
2009	4962.447979	2209.575609	0.901744	5503.169487	2.245883

In [ ]:

In [27]:
```
macro_df["Price Level"].loc[1998:2004]
```

Out[27]:
```
1998    9153.054986
1999    6720.744495
2000    1280.129259
2001    3937.372663
2002    8786.779709
2003    3497.113698
2004    5652.880900
Name: Price Level, dtype: float64
```

In [ ]:
```

```

In [28]:
```
macro_df["Price Level"].iloc[1998:2004]
```

Out[28]:
```
Series([], Name: Price Level, dtype: float64)
```

In [ ]:
```

```

In [29]:
```
macro_df.loc[1998:2004, ["Real GDP", "Money"]]
```

Out[29]:

	Real GDP	Money
1998	0.726208	2150.748276
1999	0.619796	4388.539561
2000	2.492107	7996.247273
2001	2.373067	581.357893
2002	0.547804	4814.917413
2003	1.625539	8785.949532
2004	0.449235	8627.481069

In [ ]:
```

```

In [30]:
```
macro_df[["Real GDP", "Money"]]
```

Out[30]:

	Real GDP	Money
1990	3.016167	2738.591205
1991	107.841545	2853.011501
1992	2.506835	2609.317445
1993	0.518923	6819.699511
1994	14.205873	1364.376625
1995	0.872847	8403.946530
1996	0.271548	6334.065847
1997	5.129805	1687.066104
1998	0.726208	2150.748276
1999	0.619796	4388.539561
2000	2.492107	7996.247273
2001	2.373067	581.357893
2002	0.547804	4814.917413
2003	1.625539	8785.949532
2004	0.449235	8627.481069
2005	54.647961	8469.561724
2006	0.589603	8706.915645
2007	2.196360	2889.628534
2008	0.940948	5765.731529
2009	0.901744	2209.575609

In [ ]:
```

```

In [32]:
```
import matplotlib.pyplot as plt
%matpotlib inline
macro_df.plot.line()
```

```
UsageError: Line magic function `%matpotlib` not found.
```

In [ ]:
```

```

In [ ]:
```

```

In [34]:
```
for key in macro_df:
    macro_df(key):plot.line()
    plt.show()
    plt.close()
```

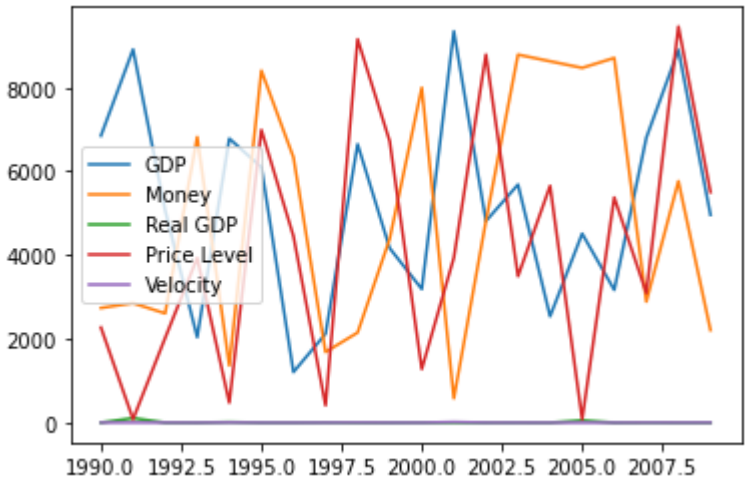
```
  File "<ipython-input-34-920dab183c42>", line 2
    macro_df(key):plot.line()
    ^
SyntaxError: illegal target for annotation
```

In [ ]:

In [ ]:

In [35]:
```
import matplotlib.pyplot as plt
%matplotlib inline
macro_df.plot.line(legend = "best")
```

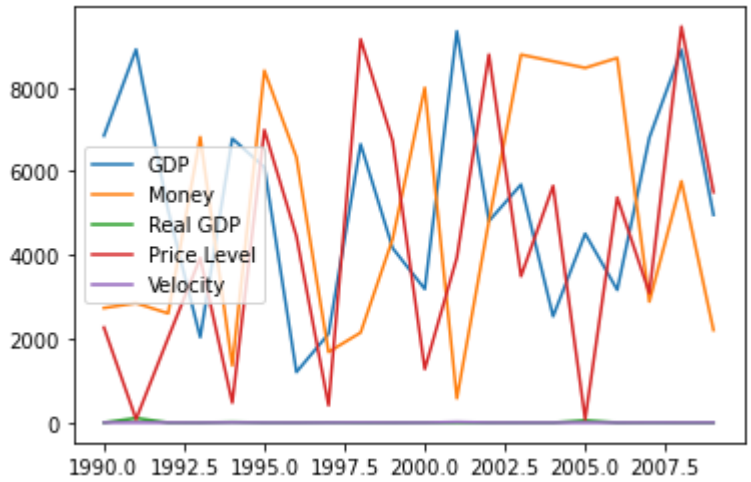
Out[35]: <AxesSubplot:>



In [ ]:

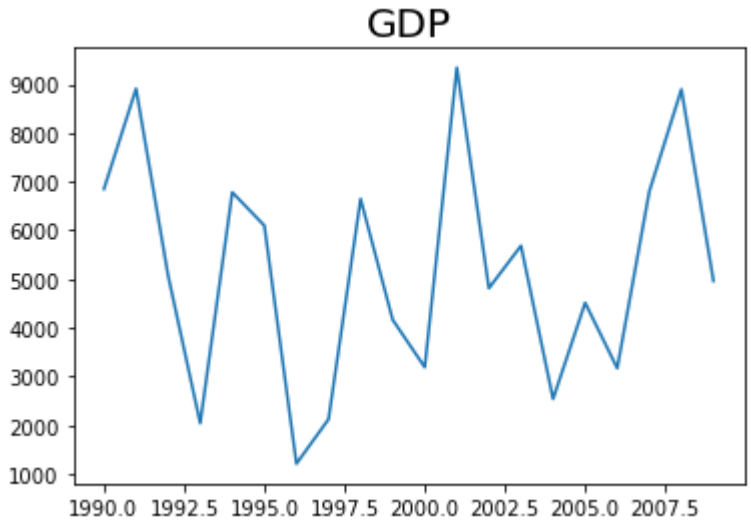
In [36]:
```
import matplotlib.pyplot as plt
%matplotlib inline
macro_df.plot.line()
```

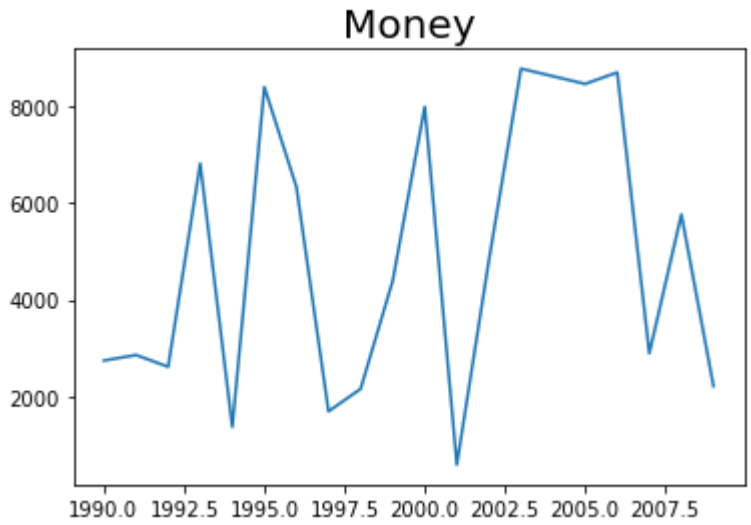
Out[36]: <AxesSubplot:>

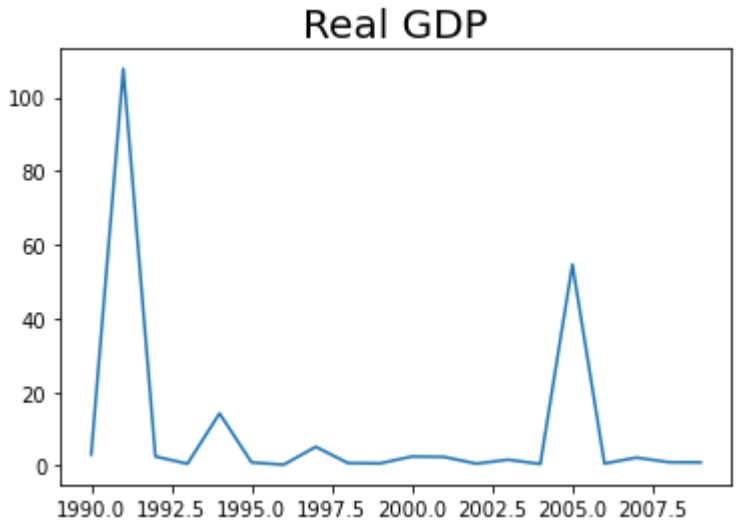


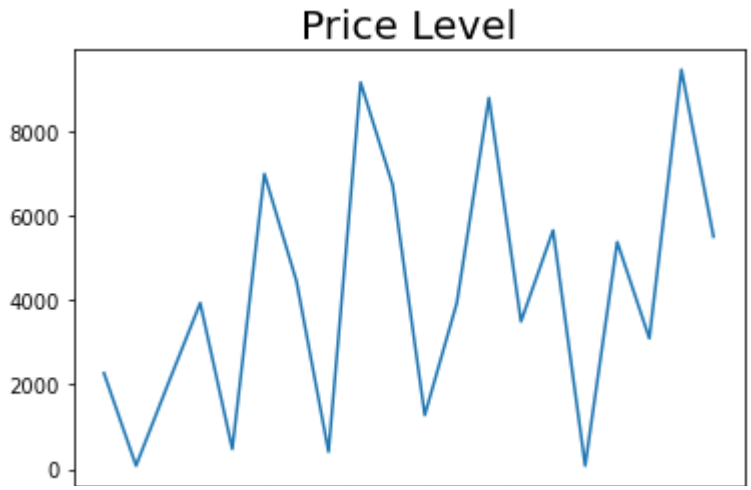
In [ ]:

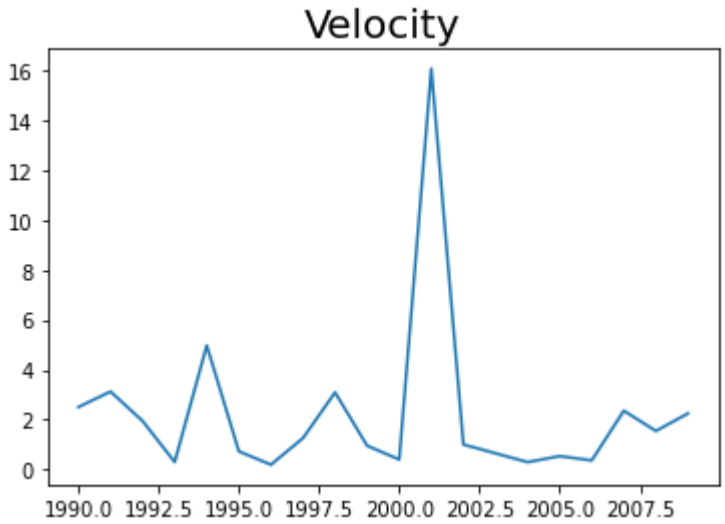
In [37]:
```
for key in macro_df:
 macro_df[key].plot.line()
 plt.title(key, fontsize = 20)
 plt.show()
 plt.close()
```





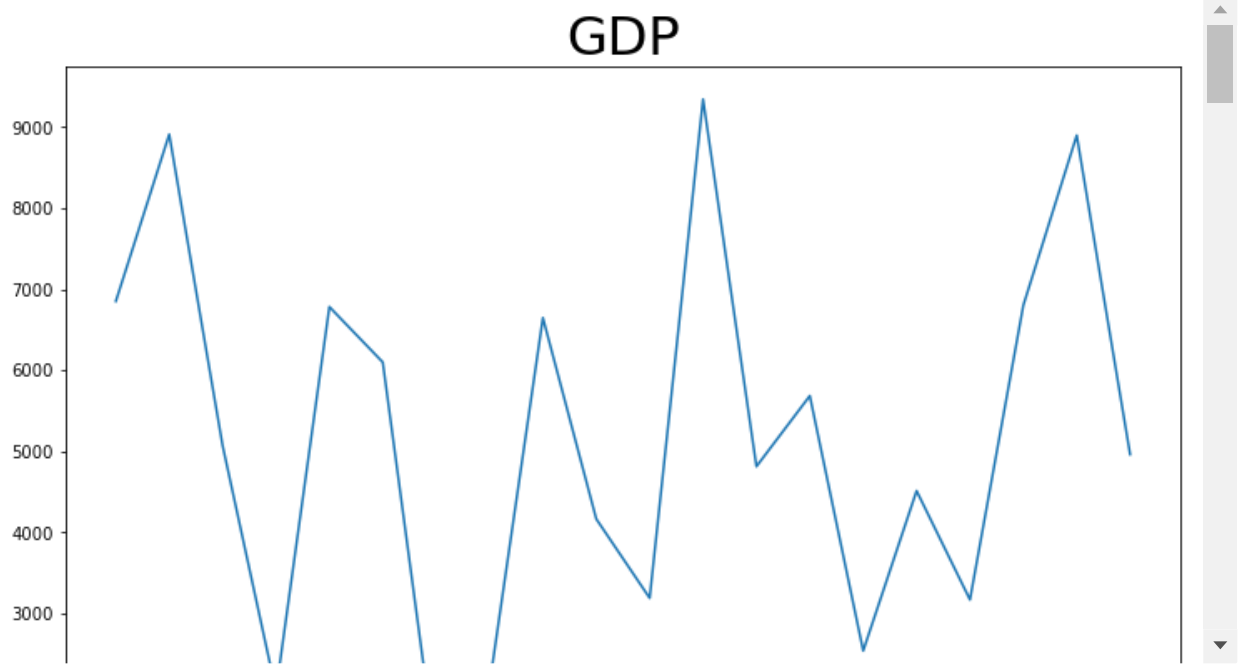



Price Level
8000
6000
4000
2000
0


Velocity
16
14
12
10
8
6
4
2
0
1990.0
1992.5
1995.0
1997.5
2000.0
2002.5
2005.0
2007.5

In [ ]:

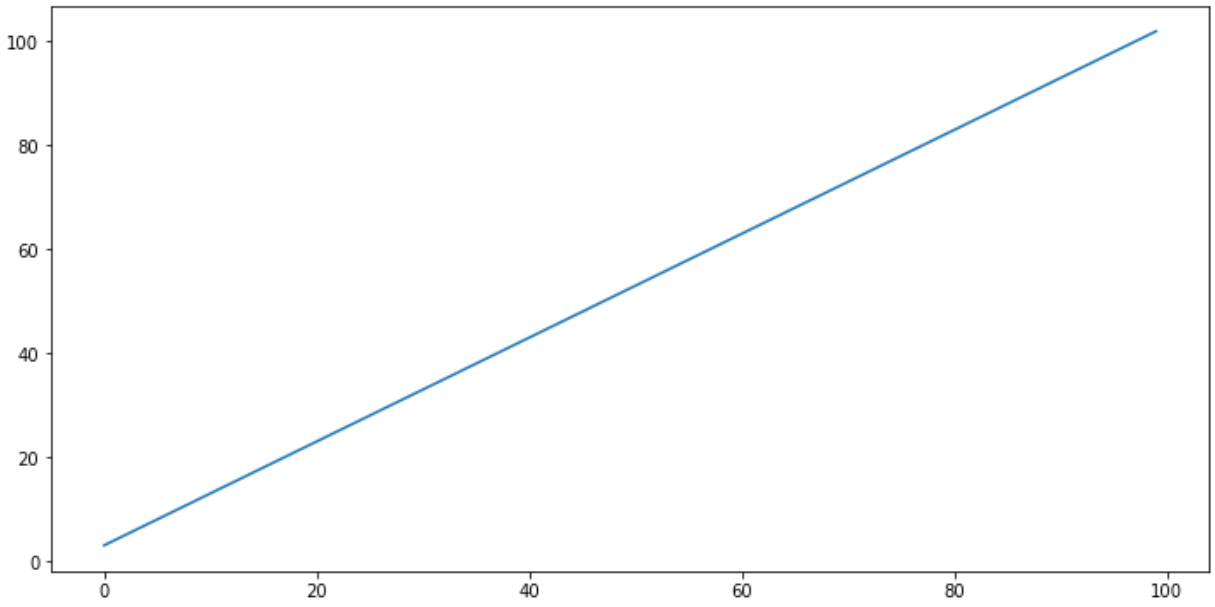
In [38]:
```
for key in macro_df:
 # fig, ax = plt.subplots() allows you to adjust different features of the plo
 fig, ax = plt.subplots(figsize = (12,8))
 macro_df[key].plot.line(ax = ax)

 # we can make the display a scatter plot if we want
 macro_df[key].plot.line(ls = "",marker = "", ax = ax)

 plt.title(key, fontsize = 32)
 plt.show()
 plt.close()
```



In [ ]:
```
for key in macro_df:
 # fig, ax = plt.subplots() allows you to adjust different features of the plot b
 fig, ax = plt.subplots(figsize = (12,8))
    macro_df[key].plot.line(color = "CO", ax = ax)

    for ix value in zip(macro_df.index)
```

In [46]:
```
line = np.array([i + 3 for i in range (100)])
figure = plt.figure(figsize = (12,6))
plt.plot(line)
```

Out[46]:
```
[<matplotlib.lines.Line2D at 0x19e8f5d6970>]
```



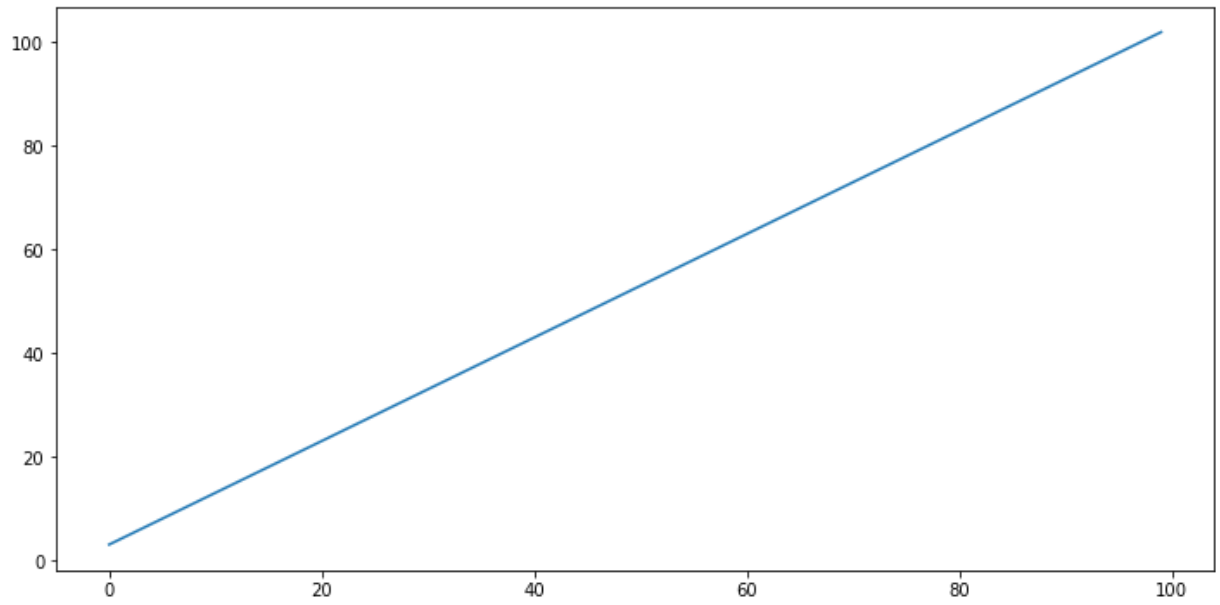
In [47]:
```
points = []
for point in line:
    points.append(random.normalvariate(point, point))
points
```

Out[47]:
```
[2.438372494066484,
 10.17581940616451,
 6.405155548590413,
 4.372372786480375,
 2.201818106715473,
 13.154201603406143,
 8.059054398350455,
 2.284762654603777,
 5.946359664732378,
 17.371310643901026,
 24.577180711605195,
 -10.112053466155515,
 19.986051111706157,
 32.714320736366105,
 9.478100127044318,
 21.65678851112834,
 10.965925738497386,
 -7.204698577206521,
 29.867369109841803,
 22.8703849568513,
 23.330566839539333,
 54.21837756387056,
 46.84995260976012,
 -14.869948201459543,
 70.45576110031169,
 45.494609054601455,
 80.67974195822751,
 11.199051688578152,
 38.43305909136018,
 38.26815224754607,
 -21.297984361094947,
 51.88868740973271,
 56.573625368158304,
 63.35809366574369,
 74.49773414200459,
 99.125033950137,
 89.17369085515155,
 29.209629659900507,
 136.03525849513554,
 20.69247867713153,
 18.644236401372726,
 14.116608431307618,
 78.03857960833594,
 66.38884399722069,
 91.62907785260998,
 34.4974764251826,
 19.015601437116903,
 114.87665402801433,
 107.09539482786153,
 26.926332560162276,
 -25.865430779636824,
 52.569049632187195,
```

```
24.26144792397924,
146.46905038247513,
46.418853319019604,
-32.53117975450705,
159.0854287418154,
44.638747705364494,
23.786670694107933,
123.60331974708129,
51.53977040685079,
162.39384508924815,
-43.353547493803816,
36.90797656179002,
133.64387088028053,
107.57707588383781,
49.28036402917431,
172.63283099504287,
269.13347443640237,
154.92986066304013,
11.696893600245836,
94.99531783260136,
92.01251598970154,
-79.28047285421269,
157.47714811459795,
-52.12700328633528,
71.13182434649765,
125.94314538133054,
107.05931401565992,
7.263480737243739,
14.644098271851206,
106.13172330317285,
165.15786381929084,
-46.54369804538973,
33.167367521643285,
-67.06928550164463,
98.03718285639704,
205.4393748389664,
169.3663277616721,
121.97332799843952,
80.64466736969052,
91.51761905751049,
86.91670814118226,
83.55547256418431,
226.93684871712426,
99.92551337419333,
40.25294878309871,
153.72848825061473,
86.02556069870042,
165.6828842036743]
```

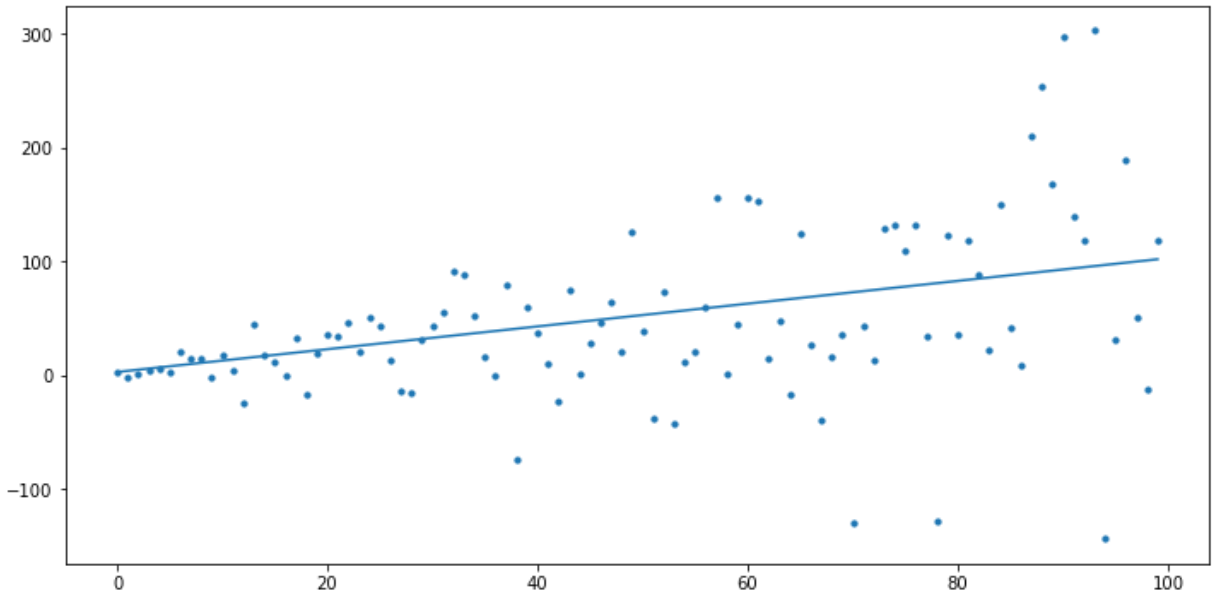
In [54]:
```
points = []
for point in line:
    points.append(random.normalvariate(point, point))

figure = plt.figure(figsize = (12,6))
plt.plot(line)
plt.scatter(x = np.arrange(len(points)), y = points, s =10)
plt.show()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-54-b3cfb787592a> in <module>
      5 figure = plt.figure(figsize = (12,6))
      6 plt.plot(line)
----> 7 plt.scatter(x = np.arrange(len(points)), y = points, s =10)
      8 plt.show()

C:\ProgramData\Anaconda3\lib\site-packages\numpy\__init__.py in __getattr__(att
r)
    301                 return Tester
    302
--> 303             raise AttributeError("module {!r} has no attribute "
    304                                  "{!r}".format(__name__, attr))
    305

AttributeError: module 'numpy' has no attribute 'arrange'
```



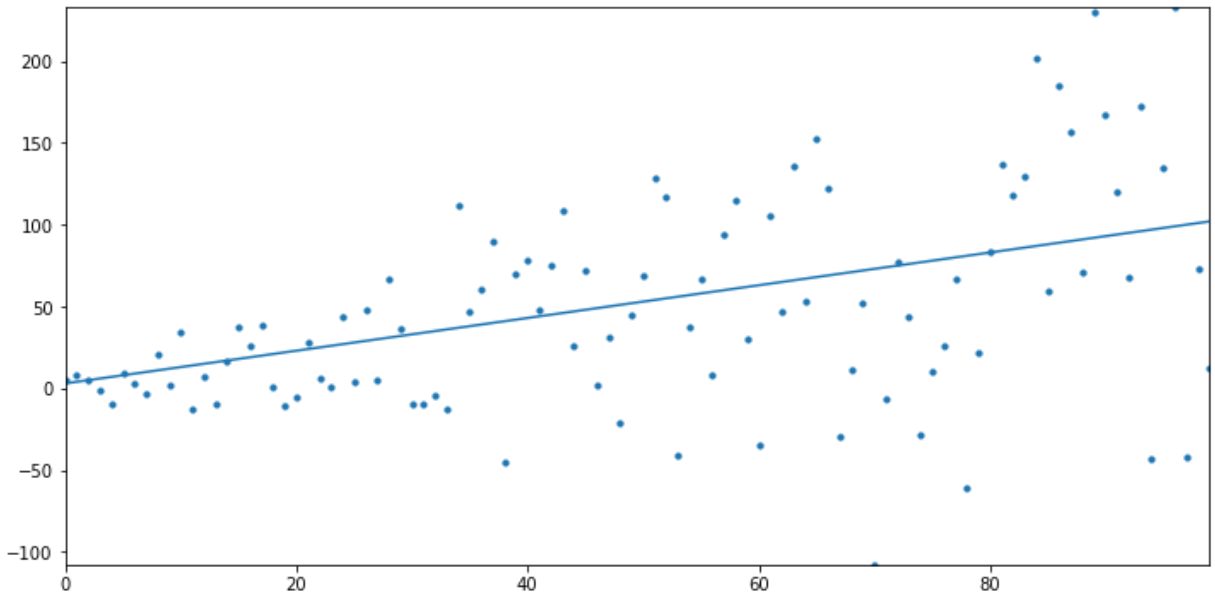
In [ ]:

In [53]:
```
points = []
for point in line:
    points.append(random.normalvariate(point, point))

figure = plt.figure(figsize = (12, 6))
plt.plot(line)
plt.scatter(np.arange(len(points)), points, s = 10)
```



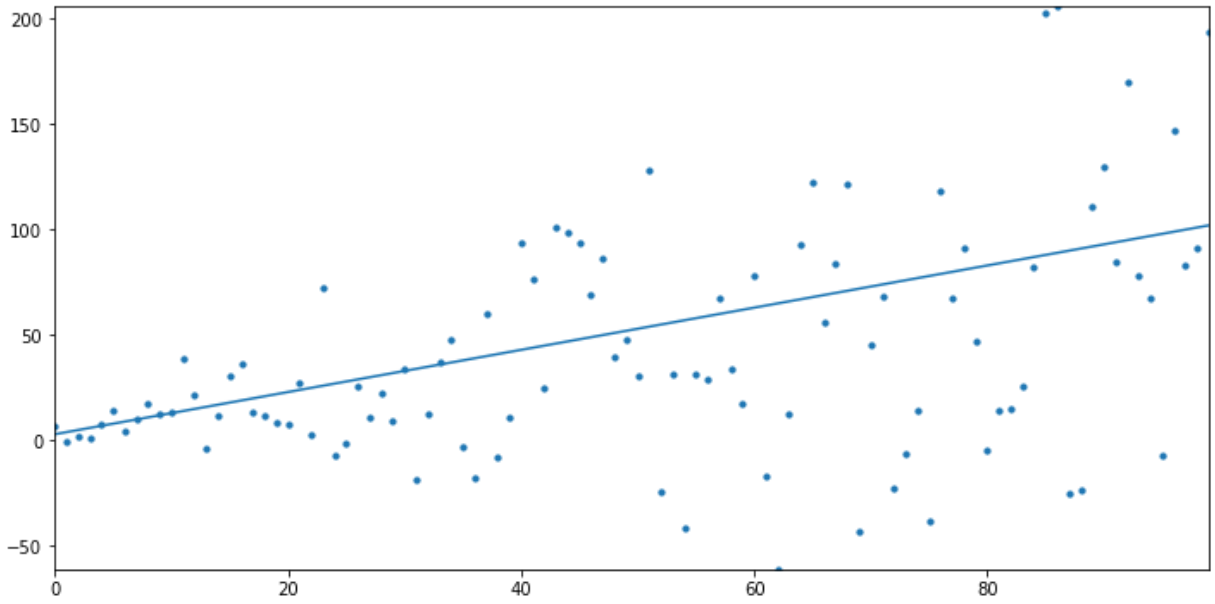
In [56]:
```
points = []
for point in line:
    points.append(random.normalvariate(point, point))

figure = plt.figure(figsize = (12, 6))
plt.plot(line)
plt.scatter(x = np.arange(len(points)), y = points, s = 10)
```

Out[56]: <matplotlib.collections.PathCollection at 0x19e902c9430>

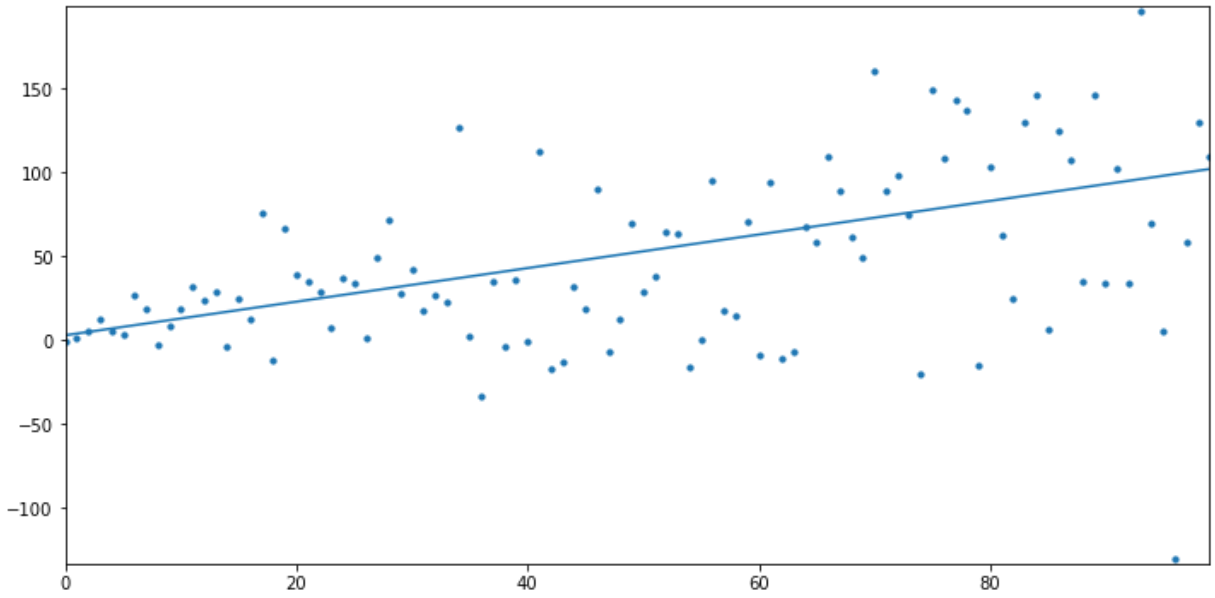


In [ ]:

In [57]:
```
points = []
for point in line:
    points.append(random.normalvariate(point, point))

figure = plt.figure(figsize = (12,6))
plt.rcParams['axes.xmargin'] = 0
plt.rcParams['axes.ymargin'] = 0
plt.plot(line)
plt.scatter(x = np.arange(len(points)), y = points, s = 10)
plt.show()
```

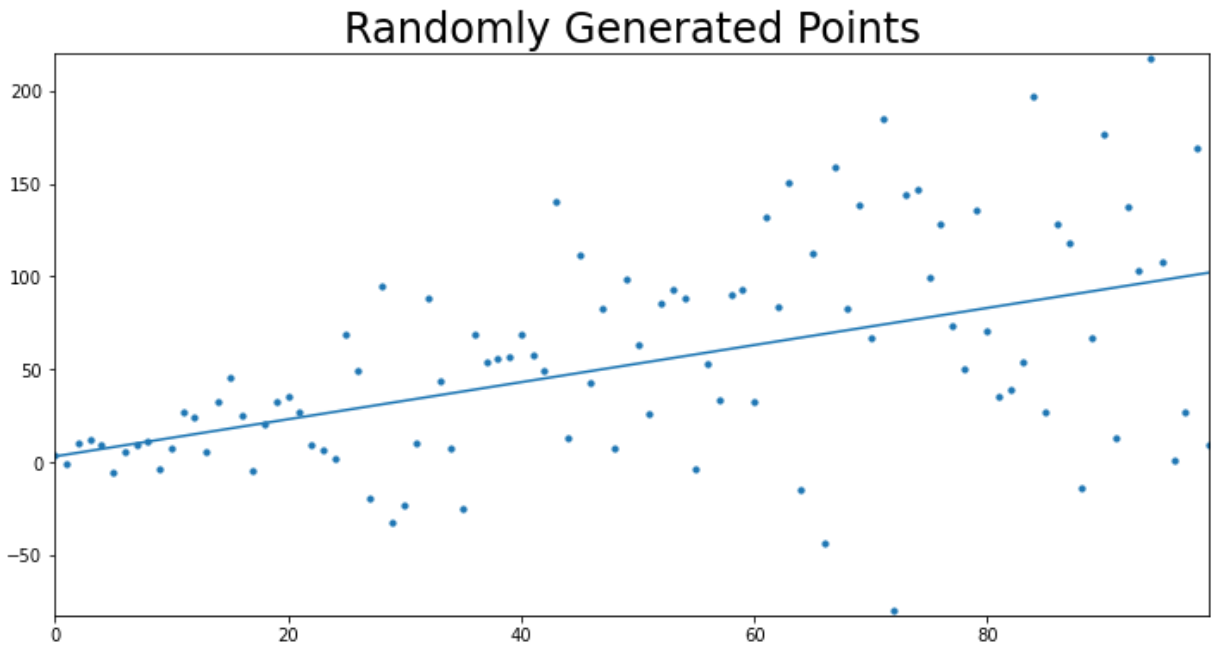


In [58]:
```
points = []
for point in line:
    points.append(random.normalvariate(point, point))

figure = plt.figure(figsize = (12,6))
plt.rcParams['axes.xmargin'] = 0
plt.rcParams['axes.ymargin'] = .01
plt.plot(line)
plt.scatter(x = np.arange(len(points)), y = points, s = 10)
plt.show()
```



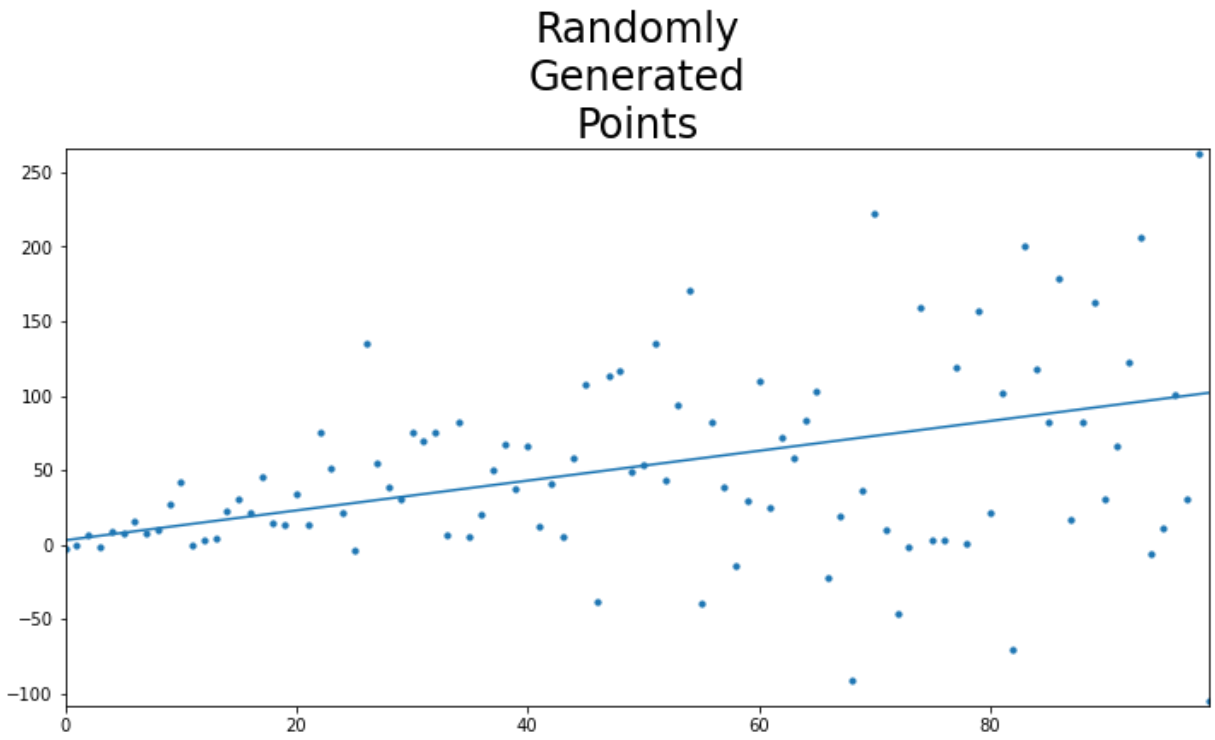
In [59]:
```
points = []
for point in line:
    points.append(random.normalvariate(point, point))

figure = plt.figure(figsize = (12,6))
plt.rcParams['axes.xmargin'] = 0
plt.rcParams['axes.ymargin'] = .01
plt.plot(line)
plt.scatter(x = np.arange(len(points)), y = points, s = 10)

plt.title("Randomly Generated Points", fontsize = 24)
```

Out[59]:
```
Text(0.5, 1.0, 'Randomly Generated Points')
```



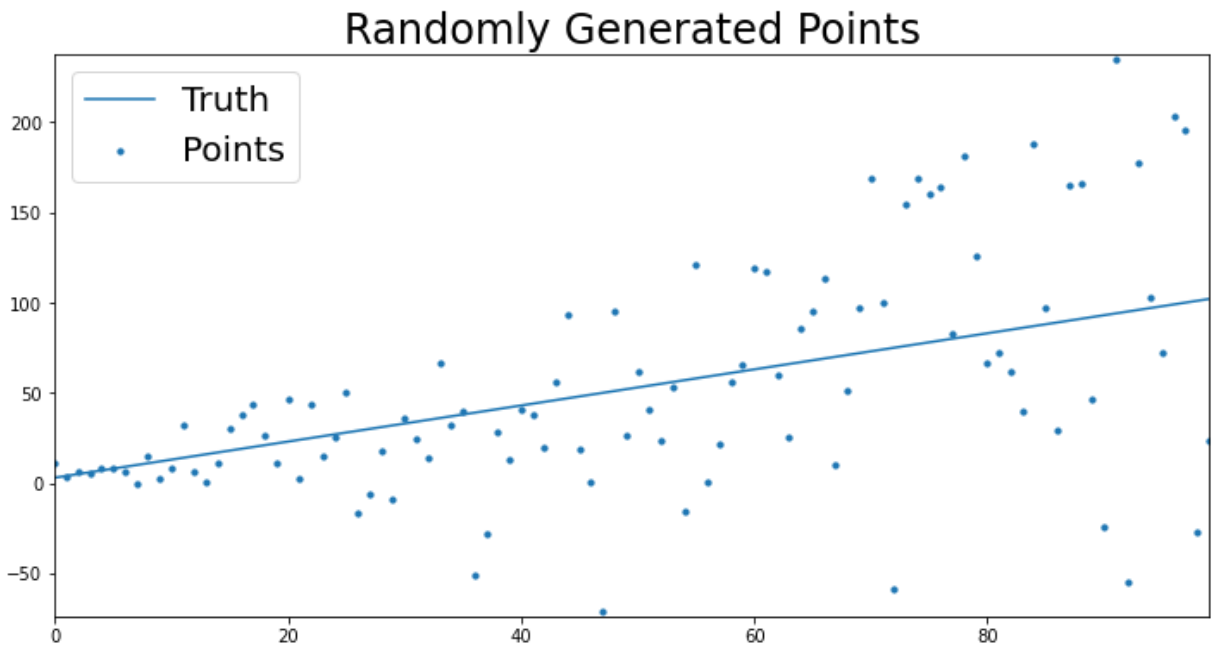
In [60]:
```
points = []
for point in line:
    points.append(random.normalvariate(point, point))

figure = plt.figure(figsize = (12,6))
plt.rcParams['axes.xmargin'] = 0
plt.rcParams['axes.ymargin'] = .01
plt.plot(line)
plt.scatter(x = np.arange(len(points)), y = points, s = 10)

plt.title("Randomly Generated Points".replace(" ","\n"), fontsize = 24)
```

Out[60]:
```
Text(0.5, 1.0, 'Randomly\nGenerated\nPoints')
```



In [63]:
```
points = []
for point in line:
    points.append(random.normalvariate(point, point))

figure = plt.figure(figsize = (12,6))
plt.rcParams['axes.xmargin'] = 0
plt.rcParams['axes.ymargin'] = .01
plt.plot(line, label = "Truth")
plt.scatter(x = np.arange(len(points)), y = points, s = 10, label = "Points")

plt.title("Randomly Generated Points", fontsize = 24)
plt.legend(fontsize = 20, loc = "best")

#Legend is basically truth and points
```

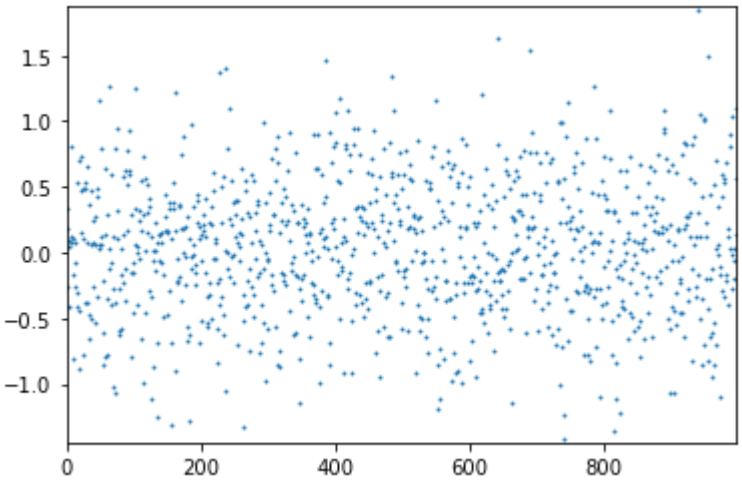
Out[63]:
```
<matplotlib.legend.Legend at 0x19e8ffb3850>
```



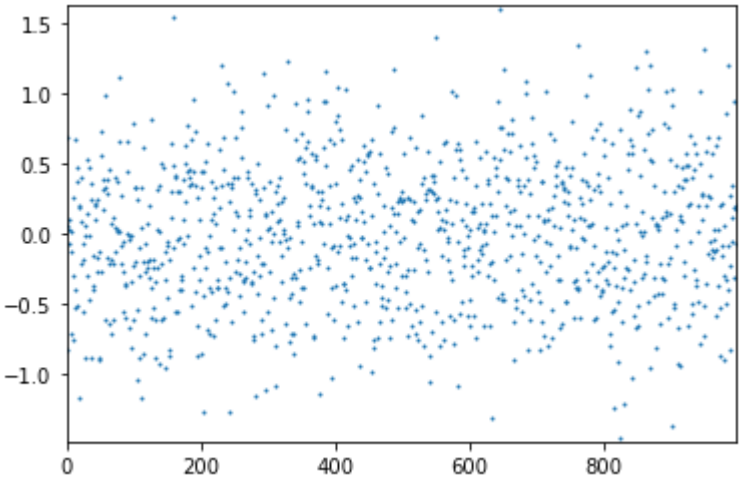
Montecarlo Simulation

1. Build list of points drawn from a standard normal distribution

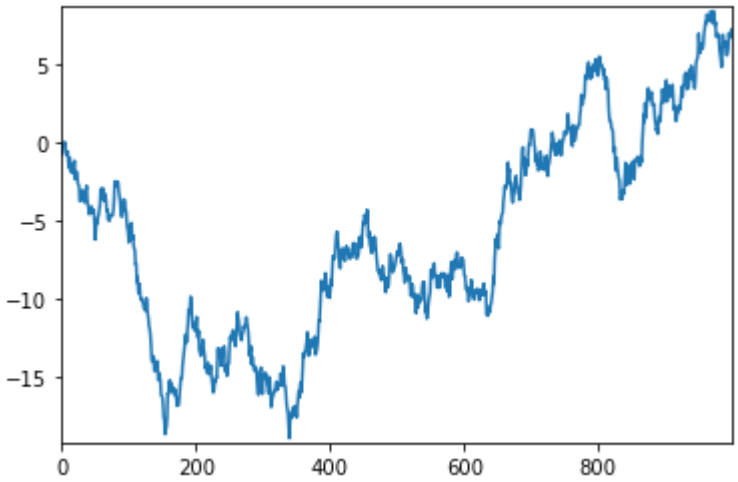
In [67]:
```
random_list = []
sim_length = 1000
for i in range(sim_length):
    random_list.append(random.normalvariate(0,.5))
plt.plot(random_list, ls = "", marker = ".", markersize = 2)
plt.show()
plt.close()
```



In [68]:
```
random_list = []
sim_length = 1000
mean = 0
sigma = 0.5


for i in range(sim_length):
    random_list.append(random.normalvariate(mean,sigma))
plt.plot(random_list, ls = "", marker = ".", markersize = 2)
plt.show()
plt.close()
```



In [69]:
```
simulation = []

for i in range(len(random_list)):
    val = random_list[i]
    if i == 0:
        simulation.append(val)
    else:
        simulation.append(simulation[i - 1] + val)

plt.plot(simulation)
plt.show()
plt.close()
```



In [70]:
```
print("line\t\t\tpoints")
list(zip(simulation, random_list))
```

```
line                    points
```

In [77]:
```
monte_carlo_sim_dict = {}
num_sims = 2000
periods = 1000
for i in range(num_sims):
    monte_carlo_sim_dict[i] = {}
    for period in range(periods):
    #create shorthand variable for current simulation
        curr_sim = monte_carlo_sim_dict[i]
        if period == 0:
            curr_sim[period] = random.normalvariate(mean, sigma)
        else:
            curr_sim[period] = curr_sim[period - 1] + random.normalvariate(mean,
```

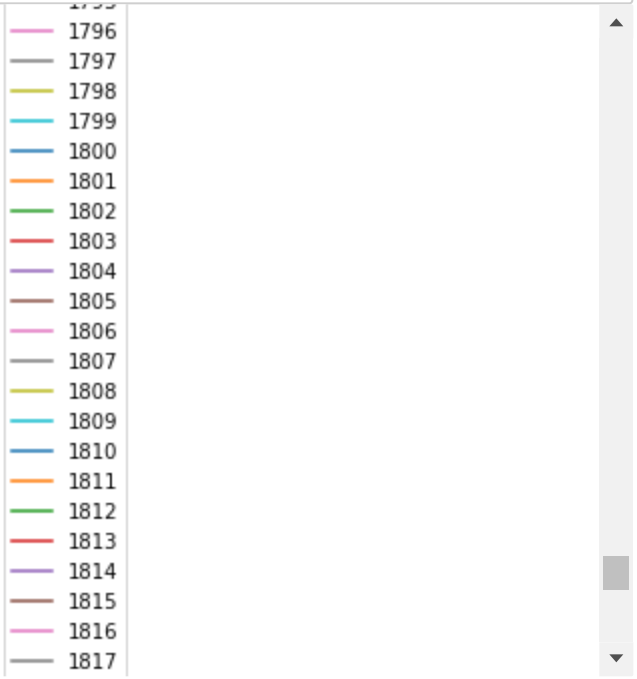
In [78]:
```
monte_carlo_sim_df = pd.DataFrame(monte_carlo_sim_dict)
monte_carlo_sim_df
```

Out[78]:

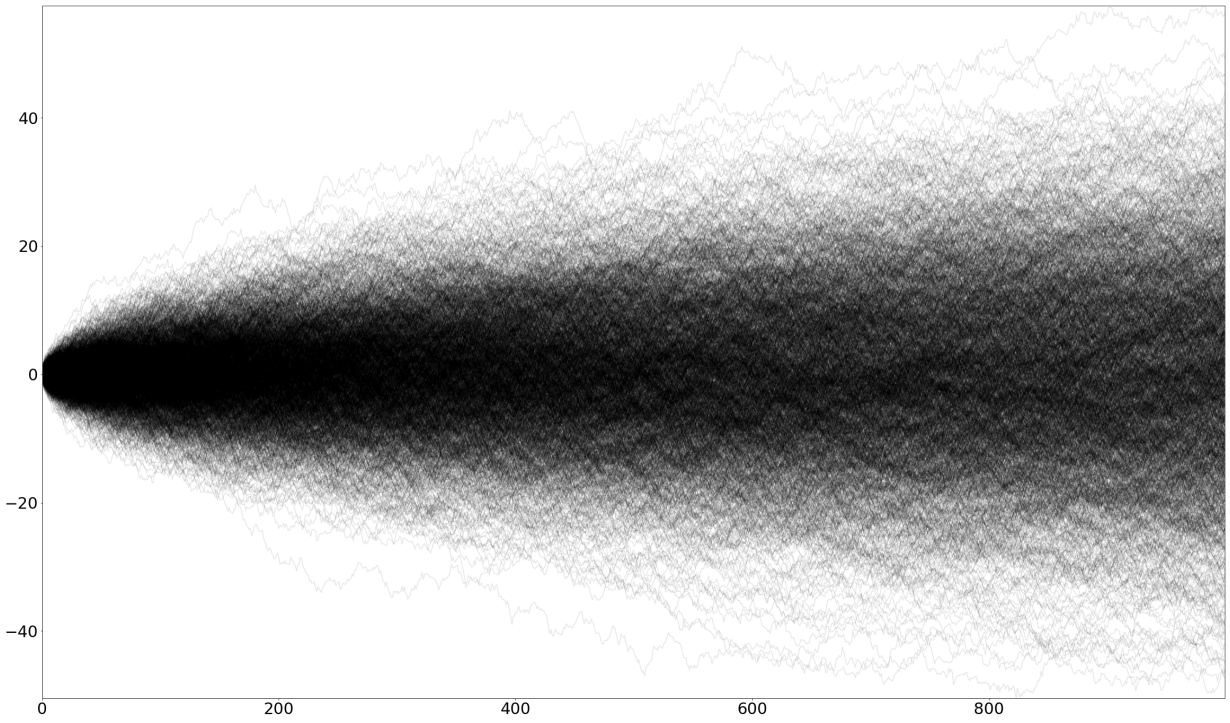
	0	1	2	3	4	5	6	7	
0	-0.271067	-0.339114	-0.901546	-0.120907	1.361958	0.454564	-0.105410	1.217615	0.0400
1	-0.637742	-1.419261	-0.968928	0.619512	1.429365	0.326242	-0.533771	0.404799	0.0654
2	-1.252357	-0.926042	-0.119360	-0.167377	1.525886	0.075108	-0.826933	0.794778	0.4115
3	-1.097197	-0.755729	0.196056	0.468419	1.993601	0.289772	-0.813750	1.038799	0.7989
4	-1.201324	-0.597090	0.597364	0.049538	2.666170	0.182846	-0.553035	1.357754	1.3708
...	...	...	...	...	...	...	...	...	
995	-6.325730	13.282495	8.161044	-16.680154	0.211931	8.758536	29.517286	8.757650	-1.9495
996	-6.195347	12.752059	9.041495	-17.433860	0.758461	10.037415	29.355008	8.109135	-1.7819
997	-5.780537	12.503204	8.857573	-17.697133	0.620691	10.084316	29.451414	8.221994	-1.5391
998	-5.903262	12.655747	8.955199	-16.623399	0.381964	10.106126	28.679344	7.968433	-1.5827
999	-5.875464	12.400744	8.852824	-16.791667	1.036276	9.221598	29.119090	7.305605	-1.9752

1000 rows × 2000 columns

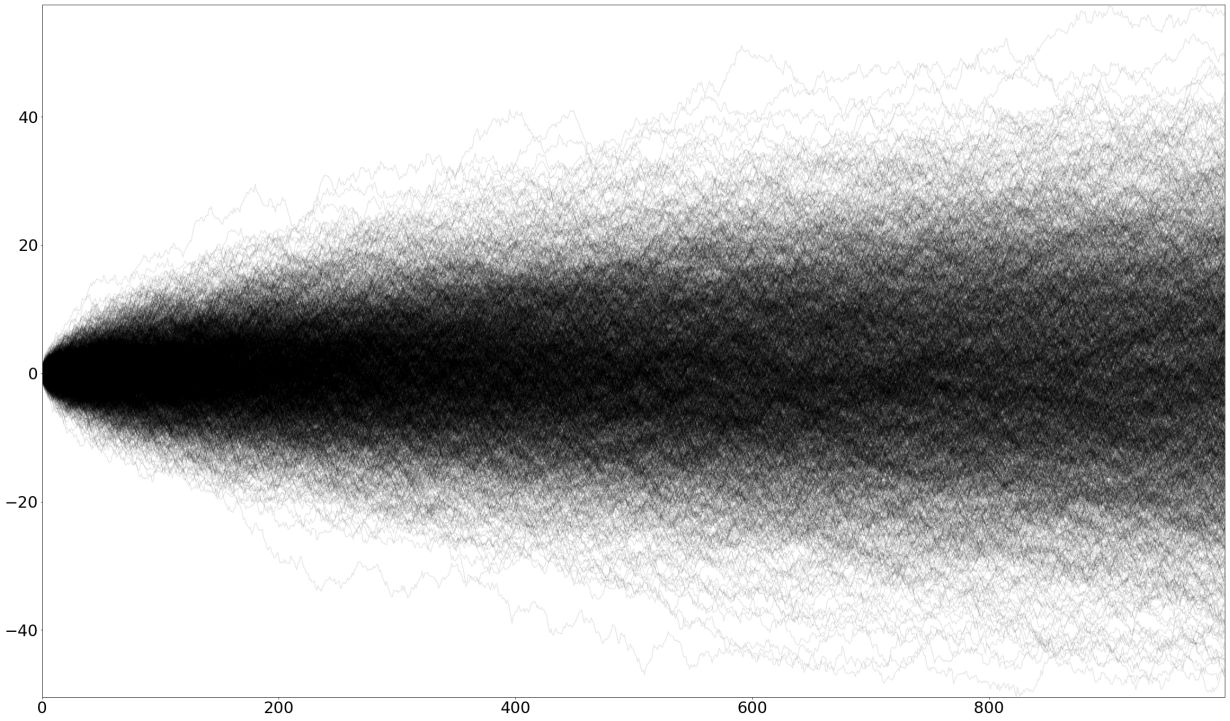
In [79]:
```
monte_carlo_sim_df.plot.line()
```



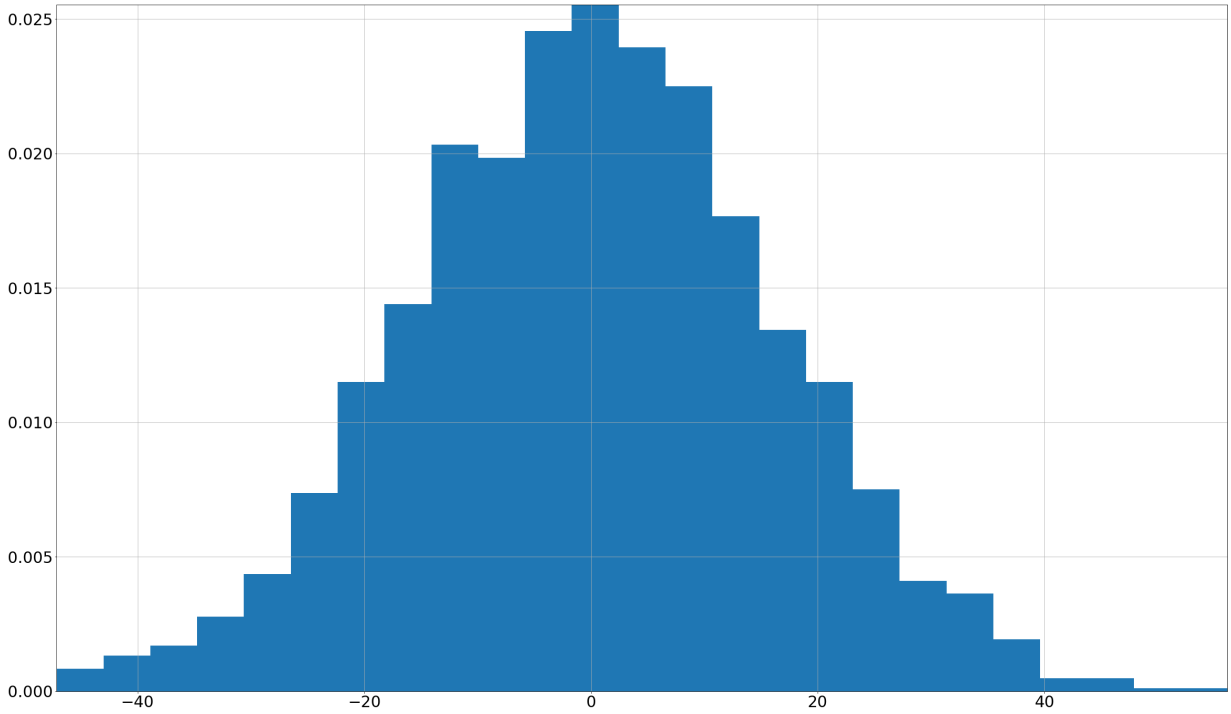
In [80]:
```
plt.rcParams['axes.xmargin'] = 0
plt.rcParams['axes.ymargin'] = 0
plt.rcParams["font.size"] = 30
fig, ax = plt.subplots(figsize = (40, 24))
monte_carlo_sim_df.plot.line(legend = False, marker = ".", markersize = .1,
                             color = "k", alpha = .1, ax = ax)
plt.show()
plt.close()
```



In [81]:
```
plt.rcParams['axes.xmargin'] = 0
plt.rcParams['axes.ymargin'] = 0
plt.rcParams["font.size"] = 30
fig, ax = plt.subplots(figsize = (40, 24))
monte_carlo_sim_df.plot.line(legend = False, marker = ".", markersize = .1,
                             color = "k", alpha = .1, ax = ax)
plt.show()
plt.close()

fig, ax = plt.subplots(figsize = (40, 24))
monte_carlo_sim_df.iloc[-1].hist(bins = 25, density = True,
                                 ax = ax)
```



Out[81]: <AxesSubplot:>



In [82]:
```
# set vertical axis values as percent
y_vals = ax.get_yticks()
ax.set_yticklabels([str(int(y * 100))+ "%" for y in y_vals])

plt.show()
plt.close()
```

```
<ipython-input-82-ea95b7771e1a>:3: UserWarning: FixedFormatter should only be u
sed together with FixedLocator
  ax.set_yticklabels([str(int(y * 100))+ "%" for y in y_vals])
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: