In [1]:
```
x = 1.5
y = 'data'
```

In [2]:
```
type(x), type(y)
```

Out[2]: (float, str)

In [3]:
```
x**2, y*2
```

Out[3]: (2.25, 'datadata')

In [8]:
```
# list
a = [x, y, 10]
a.insert(0, 'first')
a.append('last')
del a[1]
a
```

Out[8]: ['first', 'data', 10, 'last']

In [9]:
```
# tuple - constant list
b = (2, 3)
```

In [11]:
```
# dictionary - list of key-value pairs
info = {'AAPL': 130, 'META': 140}
info['META']
```

Out[11]: 140

In [12]:
```
c = [i for i in range(10)] # 0, 1,2,3,..,9
c
```

Out[12]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [17]:
```
# slice operator: [start:stop:steps]
c[:3], c[-3:], c[1:5], c[::-1], c[1:6:2]
```

Out[17]: ([0, 1, 2], [7, 8, 9], [1, 2, 3, 4], [9, 8, 7, 6, 5, 4, 3, 2, 1, 0], [1, 3, 5])

In [24]:
```
# functions
def bond_price(y, r, n):
  price = 0
  # check: n cannot be negative
  if n < 0:
    raise ValueError("Input n must be positive")
  try:
    # do execution
    if type(n) != int:
      raise TypeError("Input n must be an integer")
    for i in range(1,n+1):
      price += r/(1+y)**i
    price += 1/(1+y)**n
  except TypeError as e:
    # error is caught, do something
    print("Type error:", e)
  except ValueError as e:
    # do something else
    print("Value error:", e)
  # for i in range(n):
  #   price += r/(1+y)**(i+1)
  return price*100
```

In [25]:
```
bond_price(0.04, 0.04, 0.5)
```

```
Type error: Input n must be an integer
```

Out[25]:
```
0
```

In [26]:
```
import numpy as np
```

In [33]:
```
x = np.array(c)
z = np.zeros(shape=(2,3), dtype=int)
x.shape,
z
```

Out[33]:
```
array([[0, 0, 0],
       [0, 0, 0]])
```

In [32]:
```
# vectorization
x**2, 2*x[:4], np.exp(x[1:5])
```

Out[32]:
```
(array([ 0,  1,  4,  9, 16, 25, 36, 49, 64, 81]),
 array([0, 2, 4, 6]),
 array([ 2.71828183,  7.3890561 , 20.08553692, 54.59815003]))
```

In [34]:
```
np.exp(c)
```

Out[34]:
```
array([1.00000000e+00, 2.71828183e+00, 7.38905610e+00, 2.00855369e+01,
       5.45981500e+01, 1.48413159e+02, 4.03428793e+02, 1.09663316e+03,
       2.98095799e+03, 8.10308393e+03])
```

In [36]:
```
# c is a list
c + c # not vectorization
```

Out[36]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [37]:
```
x > 2
```

Out[37]:
```
array([False, False, False,  True,  True,  True,  True,  True,  True,
        True])
```

In [40]:
```
# broadcasting
x.shape
w1 = x.reshape((2,5))
w2 = x[:5]
w1, w2
```

Out[40]:
```
(array([[0, 1, 2, 3, 4],
        [5, 6, 7, 8, 9]]), array([0, 1, 2, 3, 4]))
```

In [41]:
```
w1 + 0.1*w2
```

Out[41]:
```
array([[0. , 1.1, 2.2, 3.3, 4.4],
       [5. , 6.1, 7.2, 8.3, 9.4]])
```

In [45]:
```
yy = np.linspace(0.0, 0.10, 101)
yy[:5]
```

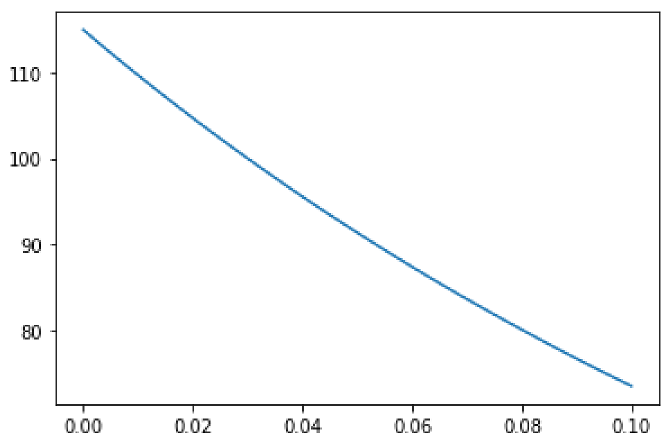
Out[45]:
```
array([0.   , 0.001, 0.002, 0.003, 0.004])
```

In [47]:
```
prices = bond_price(yy, 0.03, 5)
```

In [48]:
```
import matplotlib.pyplot as plt
```

In [49]:
```
plt.plot(yy, prices)
```

Out[49]:
```
[<matplotlib.lines.Line2D at 0x7f3bd5dd05e0>]
```



In [50]:
```
from scipy import optimize
```

In [51]:
```
def function_solve(y, target_price, r, n):
  return bond_price(y, r, n) - target_price
```

In [52]:
```
target_price = 100
r = 0.03
n = 5
optimize.newton(function_solve, 0.01, args=(target_price, r, n) )
```

Out[52]:
```
0.030000000000000013
```

In [ ]:
```
# pass by reference vs value
```