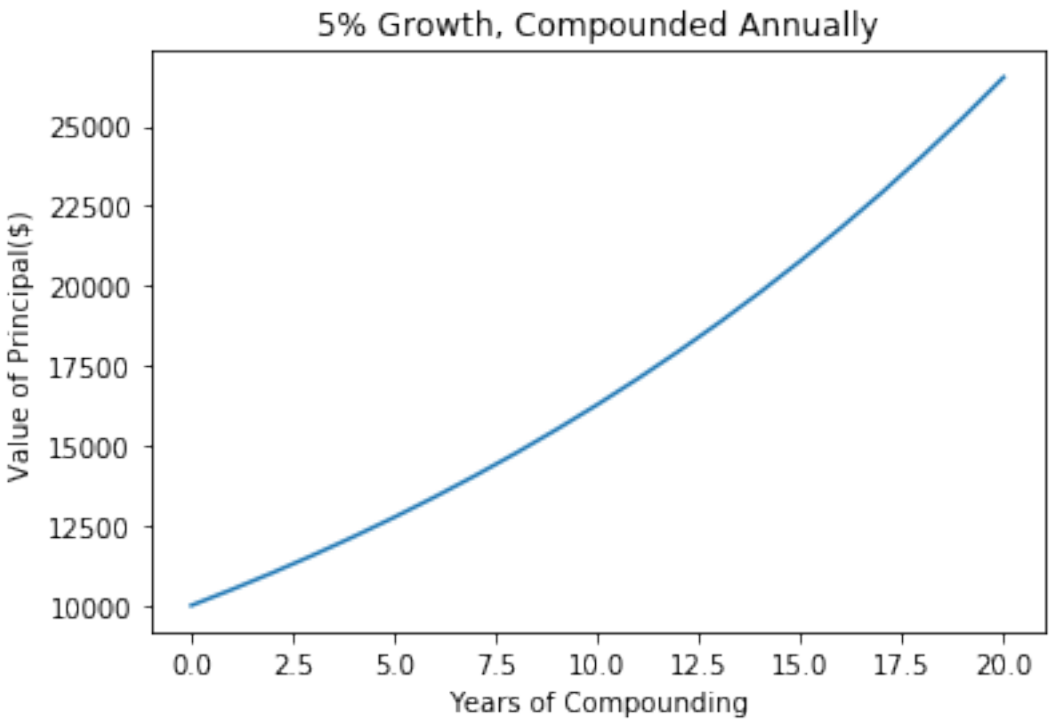
11.1 Plotting

July 29, 2018

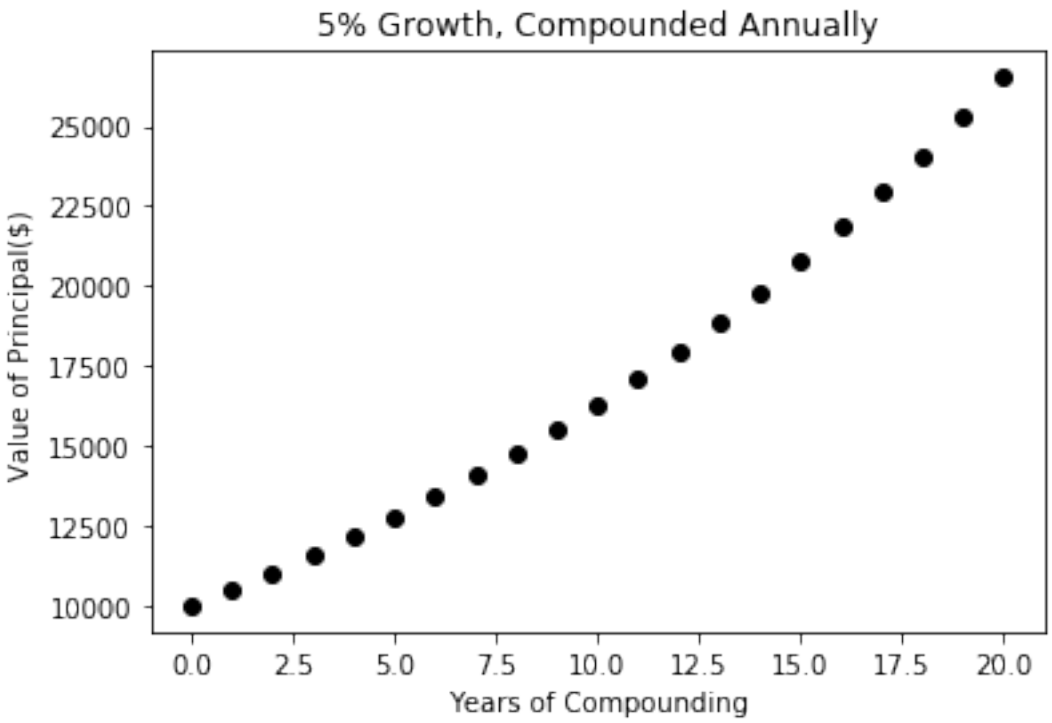
In [1]:
```
# Lab excersice 1, p. 169-175
# Alla Topp
```

In [12]:
```
import matplotlib.pyplot as plt
%matplotlib inline

principal = 10000 #initial investment
interestRate = 0.05
years = 20
values = []
for i in range(years + 1):
    values.append(principal)
    principal += principal*interestRate
pylab.plot(values)

pylab.title('5% Growth, Compounded Annually')
pylab.xlabel('Years of Compounding')
pylab.ylabel('Value of Principal($)')
```

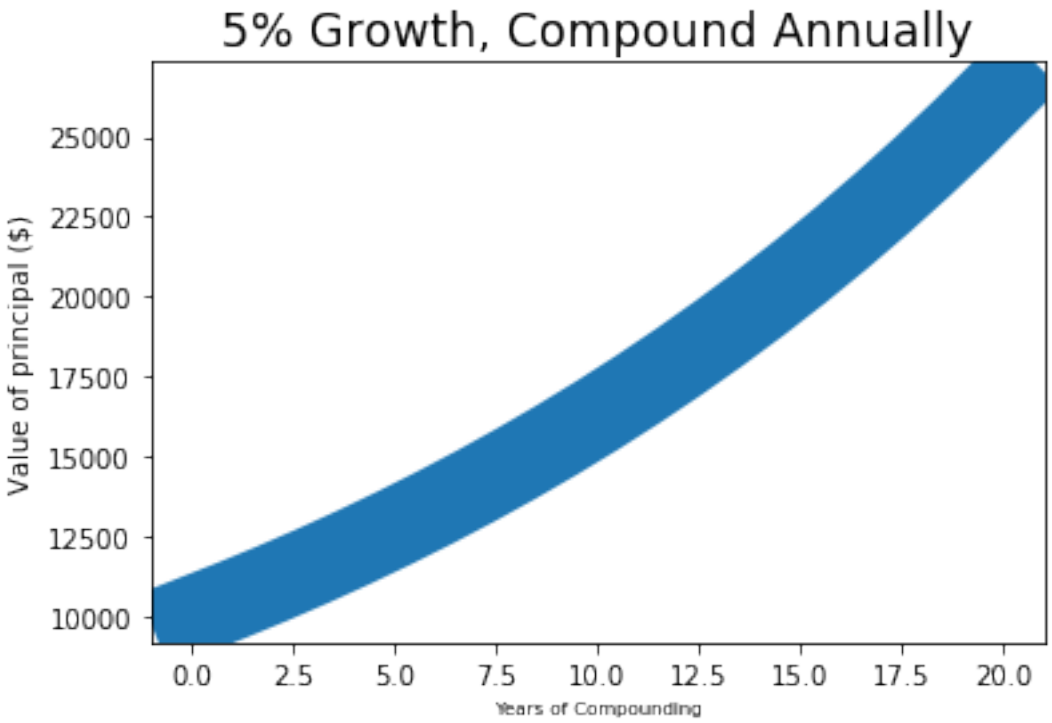
Out[12]:
```
Text(0,0.5,'Value of Principal($)')
```




```
In [13]: pylab.plot(values,'ko')
         pylab.title('5% Growth, Compounded Annually')
         pylab.xlabel('Years of Compounding')
         pylab.ylabel('Value of Principal($)')

Out[13]: Text(0,0.5,'Value of Principal($)')
```




```
In [14]: pylab.plot(values, linewidth =30)
         pylab.title('5% Growth, Compound Annually', fontsize = 'xx-large')
         pylab.xlabel('Years of Compounding', fontsize = 'x-small')
         pylab.ylabel('Value of principal ($)')

Out[14]: Text(0,0.5,'Value of principal ($)')
```



In [15]:
```
# Results:
# We can see three different graphs which show us the same results,
# but with different types of line that can be altered with some code.
# The plots show the growth of an initial investment of $10000
# at annually compounded interest rate of 5%
# The x label indicates the years of compounding, y label indicates the value of prin
# and the title contains the interest rate (5%) and the information that it compounds
```

DFS and BFS

July 28, 2018

In [3]:
```
# Finger exercise, p. 201 of the textbook
# Alla Topp
# Finding shortest path by DFS and BFS

class Node:
    def __init__(self, name):
        self.name = name
    def getName(self):
        return self.name
    def __str__(self):
        return self.name

class Edge:
    def __init__(self, src, dest):
        self.src = src
        self.dest = dest
    def getSource(self):
        return self.src
    def getDestination(self):
        return self.dest
    def __str__(self):
        return self.src.getName() + '->' + self.dest.getName()

class Weightededge(Edge):
    def __init__(self, src, dest, weight = 1.0):
        self.src = src
        self.dest = dest
        self.weight = weight
    def getWeight(self):
        return self.weight
    def __str__(self):
        return self.src.getName() +  '->('  + str(self.weight) + ')'\
               + self.dest.getName()

class Digraph:
    def __init__(self):
        self.nodes = []
        self.edges = {}
```

```
    def addNode(self, node):
        if node in self.nodes:
            raise ValueError('Duplicate node')
        else:
            self.nodes.append(node)
            self.edges[node] = []
    def addEdge(self, edge):
        src = edge.getSource()
        dest = edge.getDestination()
        if not (src in self.nodes and dest in self.nodes):
            raise ValueError('Node not in graph')
        self.edges[src].append(dest)
    def childrenOf(self, node):
        return self.edges[node]
    def hasNode(self, node):
        return node in self.nodes
    def __str__(self):
        result = ''
        for src in self.nodes:
            for dest in self.edges[src]:
                result = result + src.getName() + '->'\
                         + dest.getName() + '\n'
        return result[:-1]

class Graph(Digraph):
    def addEdge(self, edge):
        Digraph.addEdge(self, edge)
        rev = Edge(edge.getDestionation(), edge.getsource())
        Digraph.addEdge(self, rev)

def printPath(path):
    result = ''
    for i in range(len(path)):
        result = result + str(path[i])
        if i != len(path) - 1:
            result = result + '->'
    return result

def DFS(graph, start, end, path, shortest, toPrint = False):
    path = path + [start]
    if toPrint:
        print('Current DFS path:', printPath(path))
    if start == end:
        return path
    for node in graph.childrenOf(start):
        if node not in path:
            if shortest == None or len(path) < len(shortest):
                newPath = DFS(graph, node, end, path, shortest, toPrint)
```

```
                if newPath != None:
                    shortest = newPath
    return shortest

def shortestPath(graph, start, end, toPrint = True):
    return DFS(graph, start, end, [], None, toPrint)


def BFS(graph, start, end, toPrint = False):
    initPath = [start]
    pathQueue = [initPath]
    if toPrint:
        print('Current BFS path:', printPath(path))
    while len(pathQueue) != 0:
        tmpPath = pathQueue.pop(0)
        print('Current BFS path:', printPath(tmpPath))
        lastNode = tmpPath[-1]
        if lastNode == end:
            return tmpPath
        for nextNode in graph.childrenOf(lastNode):
            if nextNode not in tmpPath:
                newPath = tmpPath + [nextNode]
                pathQueue.append(newPath)
    return None

def testSP():
    nodes = []
    for name in range(6):
        nodes.append(Node(str(name)))
    g = Digraph()
    for n in nodes:
        g.addNode(n)
    g.addEdge(Edge(nodes[0], nodes[1]))
    g.addEdge(Edge(nodes[1], nodes[2]))
    g.addEdge(Edge(nodes[2], nodes[3]))
    g.addEdge(Edge(nodes[2], nodes[4]))
    g.addEdge(Edge(nodes[3], nodes[4]))
    g.addEdge(Edge(nodes[3], nodes[5]))
    g.addEdge(Edge(nodes[0], nodes[2]))
    g.addEdge(Edge(nodes[1], nodes[0]))
    g.addEdge(Edge(nodes[3], nodes[1]))
    g.addEdge(Edge(nodes[4], nodes[0]))
    sp = shortestPath(g, nodes[0], nodes[5], toPrint = True)
    print('Shortest path is' , printPath(sp))
    sp = BFS(g, nodes[0], nodes[5])
    print('Shortest path found by BFS:', printPath(sp))

testSP()
```

```
Current DFS path: 0
Current DFS path: 0->1
Current DFS path: 0->1->2
Current DFS path: 0->1->2->3
Current DFS path: 0->1->2->3->4
Current DFS path: 0->1->2->3->5
Current DFS path: 0->1->2->4
Current DFS path: 0->2
Current DFS path: 0->2->3
Current DFS path: 0->2->3->4
Current DFS path: 0->2->3->5
Current DFS path: 0->2->3->1
Current DFS path: 0->2->4
Shortest path is 0->2->3->5
Current BFS path: 0
Current BFS path: 0->1
Current BFS path: 0->2
Current BFS path: 0->1->2
Current BFS path: 0->2->3
Current BFS path: 0->2->4
Current BFS path: 0->1->2->3
Current BFS path: 0->1->2->4
Current BFS path: 0->2->3->4
Current BFS path: 0->2->3->5
Shortest path found by BFS: 0->2->3->5
```

```
In [ ]: # Is the first path found by BFS guaranteed to minimize
        # the sum of the weights of the edges?
        # We define the weight of a path as the sum of the weights
        # of the edges along that path.
        # I think that first path does not guarantee to minimize because
        # here might be multiple paths with equal weight,
        # and if so they are all shortest weighted paths.
```