In [5]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

In [6]:
```
class Circle(object):

    # Constructor
    def __init__(self, radius=3, color='blue'):
        self.radius = radius
        self.color = color

    # Method
    def add_radius(self, r):
        self.radius = self.radius + r
        return(self.radius)

    # Method
    def drawCircle(self):
        plt.gca().add_patch(plt.Circle((0, 0), radius=self.radius, fc=s
elf.color))
        plt.axis('scaled')
        plt.show()
```

In [7]:
```
RedCircle = Circle(10, 'red')
#creating an instance of a class
```

In [8]:
```
dir(RedCircle)
```

Out[8]:
```
['__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
```

```
 '__ge__',
 '__getattribute__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__module__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
 'add_radius',
 'color',
 'drawCircle',
 'radius']
```

In [10]:
```
RedCircle.color
```

Out[10]:
```
'red'
```

In [11]:
```
RedCircle.radius
```

Out[11]:
```
10
```

In [12]:
```
RedCircle.dreawCircle
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-12-4c4056a66acb> in <module>
```

```
----> 1 RedCircle.dreawCircle

AttributeError: 'Circle' object has no attribute 'dreawCircle'
```

In [13]:
```
RedCircle.radius = 1
RedCircle.radius
```

Out[13]:
```
1
```

In [14]:
```
print('Radius of object:',RedCircle.radius)
RedCircle.add_radius(2)
print('Radius of object of after applying the method add_radius(2):',Re
dCircle.radius)
RedCircle.add_radius(5)
print('Radius of object of after applying the method add_radius(5):',Re
dCircle.radius)
```

```
Radius of object: 1
Radius of object of after applying the method add_radius(2): 3
Radius of object of after applying the method add_radius(5): 8
```

In [15]:
```
class Rectangle(object):

    # Constructor
    def __init__(self, width=2, height=3, color='r'):
        self.height = height
        self.width = width
        self.color = color

    # Method
    def drawRectangle(self):
        plt.gca().add_patch(plt.Rectangle((0, 0), self.width, self.heig
ht ,fc=self.color))
        plt.axis('scaled')
        plt.show()
```

In [16]:
```
SkinnyBlueRectangle = Rectangle(2, 10, 'blue')
```

In [17]:
```
import sys

sampleMap = {'eirmod': 1,'sed': 1, 'amet': 2, 'diam': 5, 'consetetur':
1, 'labore': 1, 'tempor': 1, 'dolor': 1, 'magna': 2, 'et': 3, 'nonumy':
1, 'ipsum': 1, 'lorem': 2}

def testMsg(passed):
    if passed:
       return 'Test Passed'
    else :
       return 'Test Failed'

print("Constructor: ")
try:
    samplePassage = analysedText("Lorem ipsum dolor! diam amet, consete
tur Lorem magna. sed diam nonumy eirmod tempor. diam et labore? et diam
magna. et diam amet.")
    print(testMsg(samplePassage.fmtText == "lorem ipsum dolor diam amet
consetetur lorem magna sed diam nonumy eirmod tempor diam et labore et
 diam magna et diam amet"))
except:
    print("Error detected. Recheck your function " )
print("freqAll: ")
try:
    wordMap = samplePassage.freqAll()
    print(testMsg(wordMap==sampleMap))
except:
    print("Error detected. Recheck your function " )
print("freqOf: ")
try:
    passed = True
    for word in sampleMap:
        if samplePassage.freqOf(word) != sampleMap[word]:
            passed = False
            break
    print(testMsg(passed))

except:
    print("Error detected. Recheck your function  " )
```

```
Constructor:
Error detected. Recheck your function

freqAll:
Error detected. Recheck your function
freqOf:
Error detected. Recheck your function
```

In [18]:

```
class analysedText(object):

    def __init__ (self, text):
        # remove punctuation
        formattedText = text.replace('.','').replace('!','').replace(
'?','').replace(',','')

        # make text lowercase
        formattedText = formattedText.lower()

        self.fmtText = formattedText

    def freqAll(self):
        # split text into words
        wordList = self.fmtText.split(' ')

        # Create dictionary
        freqMap = {}
        for word in set(wordList): # use set to remove duplicates in li
st
            freqMap[word] = wordList.count(word)

        return freqMap

    def freqOf(self,word):
        # get frequency map
        freqDict = self.freqAll()

        if word in freqDict:
            return freqDict[word]
        else:
```

```
            return 0
```

In [ ]: