This is to solve the SageMath portion of Chapter 1 of

"A journey through the realm of numbers:

from quadratic equations to quadratic reciprocity."

This text was created using a local installation of SageMath, but portions could as well be used on the single cell server https://sagecell.sagemath.org (https://sagecell.sagemath.org)

Below we follow a random trial and error path of learning SageMath and practising the material of Chapter 1 of the above text. You may try to read this as a dialog between us and SageMath, but it is advisable to not just read the dialog but to program oneself.

Disclaimer: the authors do not claim to be experts in SageMath and also not that this introduction is the best way to learn SageMath. Instead our focus remains to practice the ideas of the above book.

I. SageMath as a Calculator

In [1]:
```
1 + 2       # Shift-Enter will make sage calculate the answer
```

Out[1]: 3

In [2]:
```
2 ^ 10      # all after "#" is a comment and is ignored by sage
```

Out[2]: 1024

In [3]:
```
15 / 6      # sage will output a simplified rational number again
```

Out[3]: 5/2

In [4]:
```
i ^ 2       # we do not have to introduce sage to complex numbers
```

Out[4]: -1

In [5]:
```
(1+i) ^ 4
```

Out[5]: -4

In [6]:
```
# This 'calculates' the square root of -1 but sage makes a choice.
sqrt(-1)
```

Out[6]: I

In [7]:
```
exp(2)      # sage will not approximate the answer
```

Out[7]:
```
e^2
```

In [8]:
```
# sage knows this precisely and will not approximate the answer.
sin(pi/5)
```

Out[8]:
```
1/4*sqrt(-2*sqrt(5) + 10)
```

In [9]:
```
# The following cryptic command improves the readability
# of formulas. Note that this line is only needed once.
%display latex
sin(pi/5)      # now the output is more readable
```

Out[9]:
$$\frac{1}{4}\sqrt{-2\sqrt{5}+10}$$

In [10]:
```
sin(pi/5).n()  # this gives us a numerical approximation
```

Out[10]:
```
0.587785252292473
```

In [11]:
```
exp(2.0)       # a decimal input leads to decimal answer
```

Out[11]:
```
7.38905609893065
```

In [12]:
```
# The following gives 10000 decimal digits of pi by scrolling to
# the right (not supported in the pdf).
pi.n(digits=10000)
```

Out[12]:
```
3.14159265358979323846264338327950288419716939937510582097494459230
```

In [13]:
```
# This assignment stores the output under a variable name.
myfirstnumber = 5/2 + (1+i)^4 + sin(pi/5)
```

In [14]:
```
# The variable we can use for further calculations.
mysecondnumber = exp(myfirstnumber)
```

In [15]:
```
mysecondnumber.n()  # outputs the numerical approximation
```

Out[15]:
```
0.401633720915105
```

In [16]:
```
# The command print is especially important on SageMathCell -- as
# it otherwise only shows the result of the last calculation.
print(myfirstnumber.n(), mysecondnumber.n())
```

```
-0.912214747707527 0.401633720915105
```

In [17]:
```
i = 1       # overwrites the standard meaning of i for sage
```

In [18]:
```
i           # as we can see here
```

Out[18]:
```
1
```

In [19]:
```
reset("i") # restores its original meaning
```

In [20]:
```
i^2        # as we can see here
```

Out[20]: -1

In [21]:
```
type(1)    # in the background sage keeps track of types of numbers
```

Out[21]:
```
<class 'sage.rings.integer.Integer'>
```

In [22]:
```
type(1/2)  # normally we can ignore this
```

Out[22]:
```
<class 'sage.rings.rational.Rational'>
```

In [23]:
```
type(1.0)  # occasionally this is useful
```

Out[23]:
```
<class 'sage.rings.real_mpfr.RealLiteral'>
```

In [24]:
```
type(i)    # e.g. when tracking down an error message
```

Out[24]:
```
<class 'sage.symbolic.expression.Expression'>
```

In [25]:
```
# Entering "arc" followed by the Tab-key gives you a list of
# commands that start with "arc" (not shown in the pdf). To get
# information on a command you can enter the command followed
# by "?" (also not in the pdf).
sin?
```

II. First scripts

In [26]:
```
type(phi)
```

```
---------------------------------------------------------------------
---------
NameError                                 Traceback (most recent c
all last)
<ipython-input-26-b507834ec6c1> in <module>()
----> 1 type(phi)

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

In [27]:
```
# The following defines a symbolic function but also makes phi a
# variable name.
cis(phi) = cos(phi) + i*sin(phi)
```

In [28]:
```
type(phi)
```

Out[28]:
```
<class 'sage.symbolic.expression.Expression'>
```

In [29]:
```
i sin(pi)     # this reates a syntax error, hence * is important
```

```
  File "<ipython-input-29-ead5406e1912>", line 1
    i sin(pi)     # this reates a syntax error, hence * is importa
nt
          ^
SyntaxError: invalid syntax
```

In [30]:
```
cis(2 * pi / 3)   # this gives a third root of unity
```

Out[30]: $\frac{1}{2}i\sqrt{3} - \frac{1}{2}$

In [31]:
```
cis(2*pi/3) ^ 3   # verifying
```

Out[31]: $\left(\frac{1}{2}i\sqrt{3} - \frac{1}{2}\right)^3$

In [32]:
```
expand(cis(2*pi/3) ^ 3) # to expand the symbolic expression
```

Out[32]: 1

In [33]:
```
cis(-1.0 * i)   # this reveals a connection to Euler number e
```

Out[33]: 2.71828182845905

In [34]:
```
cis(-i)         # this reveals a connection to sinh, cosh
```

Out[34]: $\cosh(1) + \sinh(1)$

In [35]:
```
# The following defines a new function that works quite differently
# than the above symbolic cis-routine.
def midnight(p, q):
    rt = sqrt(p^2/4 - q)
    return [-p/2 + rt, -p/2 - rt]

# The definition using def needs to end with an empty line.
# Also note that [a,b] creates a list consisting of a and b.
midnight(-1, -1)   # this solves the equation x^2-x-1=0
```

Out[35]: $\left[\frac{1}{2}\sqrt{5} + \frac{1}{2}, -\frac{1}{2}\sqrt{5} + \frac{1}{2}\right]$

In [36]:
```
type(cis)          # a symbolic function
```

Out[36]:
```
<class 'sage.symbolic.expression.Expression'>
```

In [37]:
```
print(cis)         # for displaying it
cis                # automatically displayed if used without input
```

```
phi |--> cos(phi) + I*sin(phi)
```

Out[37]: $\phi \mapsto \cos(\phi) + i\,\sin(\phi)$

In [38]:
```
derivative(cis)    # an advantage
```

Out[38]: $\phi \mapsto i\,\cos(\phi) - \sin(\phi)$

In [39]:
```
type(midnight)     # a quite different type
```

Out[39]: <class 'function'>

In [40]:
```
print(midnight)    # this produces a meaningless output
midnight
```

```
<function midnight at 0x245035268>
```

Out[40]: <function midnight at 0x245035268>

In [41]:
```
derivative(midnight) # this makes no sense to sage
```

```
---------------------------------------------------------------------
---------
TypeError                                 Traceback (most recent c
all last)
<ipython-input-41-041e31b362a3> in <module>()
----> 1 derivative(midnight) # this makes no sense to sage

/Users/manfred/Documents/SageMath9/local/lib/python3.7/site-packag
es/sage/calculus/functional.py in derivative(f, *args, **kwds)
    133         pass
    134     if not isinstance(f, Expression):
--> 135         f = SR(f)
    136     return f.derivative(*args, **kwds)
    137

/Users/manfred/Documents/SageMath9/local/lib/python3.7/site-packag
es/sage/structure/parent.pyx in sage.structure.parent.Parent.__cal
l__ (build/cythonized/sage/structure/parent.c:9218)()
    898         if mor is not None:
    899             if no_extra_args:
--> 900                 return mor._call_(x)
    901             else:
    902                 return mor._call_with_args(x, args, kwds)

/Users/manfred/Documents/SageMath9/local/lib/python3.7/site-packag
es/sage/structure/coerce_maps.pyx in sage.structure.coerce_maps.De
faultConvertMap_unique._call_ (build/cythonized/sage/structure/coe
rce_maps.c:4556)()
    159                 print(type(C), C)
    160                 print(type(C._element_constructor), C._ele
ment_constructor)
--> 161             raise
    162
    163     cpdef Element _call_with_args(self, x, args=(), kwds={
}):

/Users/manfred/Documents/SageMath9/local/lib/python3.7/site-packag
es/sage/structure/coerce_maps.pyx in sage.structure.coerce_maps.De
faultConvertMap_unique._call_ (build/cythonized/sage/structure/coe
```

```
rce_maps.c:4448)()
    154         cdef Parent C = self._codomain
    155         try:
--> 156             return C._element_constructor(x)
    157         except Exception:
    158             if print_warnings:

/Users/manfred/Documents/SageMath9/local/lib/python3.7/site-packag
es/sage/symbolic/ring.pyx in sage.symbolic.ring.SymbolicRing._elem
ent_constructor_ (build/cythonized/sage/symbolic/ring.cpp:7011)()
    386             return prod([SR(p)**e for p,e in x], SR(x.unit
()))
    387         else:
--> 388             raise TypeError(f"unable to convert {x!r} to a
symbolic expression")
    389
    390         return new_Expression_from_GEx(self, exp)

TypeError: unable to convert <function midnight at 0x245035268> to
a symbolic expression
```

In [42]:
```
# To see why the def-construction is useful:
(2 / 3).denominator()  # this calculates the denominator
```

Out[42]: 3

In [43]:
```
# The following two lines should calculate the denominator of 2/3,
# but this does not work.
denom(r) = r.denominator()
denom(2 / 3)
```

Out[43]: 1

In [44]:
```
# To see why the above does not work, we verify types and values
# of various expressions involved. This reveals that Sage
# calculated the denominator of the symbolic expression r.
print(denom)
print(type(r))
print(r.denominator())
print((r / phi).denominator())
```

```
r |--> 1
<class 'sage.symbolic.expression.Expression'>
1
phi
```

In [45]:
```
def denom(r):
    return r.denominator()

# The following now works correctly, which suggests that we should
# use the def-construction (unless we have reasons to work with
# the symbolic definition).
denom(2 / 3)
```

Out[45]: 3

In [46]:
```
# In the following example we use the if-clause together with the
# not-equal operator !=. The else-if clause and the else-clause
# are optional, but it is important to use indentation.
def midnight3(a, b, c):       # solves the equation ax^2+bx+c=0
    if a != 0:
        return midnight(b/a, c/a)
    elif b != 0:
        return [-c/b]
    else:
        print("The equation makes no sense.")
```

In [47]:
```
midnight3(1, -1, -1) # this calculates the solution to x^2-x-1=0
```

Out[47]:
$$\left[\frac{1}{2}\sqrt{5}+\frac{1}{2}, -\frac{1}{2}\sqrt{5}+\frac{1}{2}\right]$$

In [48]:
```
midnight3(0, 1, -1)  # the equation 0x^2+x-1=0 instead
```

Out[48]:
$[1]$

In [49]:
```
midnight3(0, 0, 1)
```

```
The equation makes no sense.
```

In [50]:
```
# The following calculates the argument of z.
def angle(z):
    x=z.real().n()
    y=z.imag().n()
    if x>=0 and y>=0:
        return arctan(y / x)           # first quadrant
    elif x<0 and y>=0:
        return arctan(y/x) + pi.n()    # second quadrant
    elif x<0 and y<0:
        return arctan(y/x) + pi.n()    # third quadrant
    else:
        return arctan(y/x) + 2*pi.n() # fourth quadrant
```

In [51]:
```
# The following calculates the absolute value of z.
def absval(z):
    return sqrt(z.real()^2 + z.imag()^2).n()
```

In [52]:
```
# The following calculates all three cube roots of w.
def cbrt(w):
  if w == 0:
    return [0, 0, 0]
  else:
    az = absval(w) ^ (1/3)
    phi = angle(w) / 3
    return [az*cis(phi).n(digits=3),\
            az*cis(phi + 2*pi/3).n(digits=3),
            az*cis(phi + 4*pi/3).n(digits=3)]
# If a command is too long to fit in one line, one can end
# the command in the middle with a single backslash and for
# Sage the line will continue on the next line. If a
# parenthesis is open the backslash is not necessary,
# but it is good practice to have consistent
# self-explanatory indentation in such cases.
```

In [53]:
```
# The following example shows the disadvantage of using numerical
# approximations too early. The same problem will again be visible
# in other examples below.
cbrt(i)
```

Out[53]: $[0.866 + 0.500i, -0.866 + 0.500i, 0.0000134 - 1.00i]$

III. Lists & Loops

In [54]:
```
v=["a", "b", "c"]  # creates a list
```

In [55]:
```
v[1]               # this reads out the second entry
```

Out[55]: b

In [56]:
```
[1 .. 5]           # a numerical list with start and end point
```

Out[56]: $[1, 2, 3, 4, 5]$

In [57]:
```
range(5)           # range creates a template for a list
```

Out[57]: range(0, 5)

In [58]:
```
range(1 ,5)        # starting at 1 instead
```

Out[58]: range(1, 5)

In [59]:
```
range(1, 10, 2)    # going by steps of 2s
```

Out[59]: range(1, 10, 2)

In [60]:
```
# The following for-construction is very important.
for text in ["hello", "world", "!"]:
    print(text)
for n in range(3):
    print(n)
```

```
hello
world
!
0
1
2
```

In [61]:
```
# If we change the indentation in the above examples we get
# a different output. I.e. the indentation indicates whether
# the first for-construction is finished and we have a second
# independent one (like above), or whether we have the second
# as part of the instructions of the first.
for text in ["hello", "world", "!"]:
    print(text)
    for n in range(3):
        print(n)
```

```
hello
0
1
2
world
0
1
2
!
0
1
2
```

In [62]:
```
# The following is useful shortcut for creating lists.
[n^2 for n in range(5)]
```

Out[62]: $[0, 1, 4, 9, 16]$

In [63]:
```
# The following calculates the solutions to x^3+px+q=0.
def redcubic(p,q):
    if p == 0:                          # special case
       return cbrt(-q)
    else:
       u = midnight(q, -(p/3)^3)[0]     # only first solution needed
       zulist = cbrt(u)                 # list of all cube roots
       return [zu - p/(3*zu) for zu in zulist] # same shortcut
```

In [64]:
```
redcubic(0,1)              # testing special case x^3+1=0
```

Out[64]: $\left[0.500 + 0.866i, -1.00 - 8.91 \times 10^{-6} i, 0.500 - 0.866i\right]$

In [65]:
```
expand((x-3) * (x-5) * (x+8)) # creates an example
```

Out[65]: $x^3 - 49\,x + 120$

In [66]:
```
redcubic(-49, 120)          # testing a less special case
```

Out[66]: $[5.00 + 0.000244i, -8.00, 3.00]$

In [67]:
```
# In the following we use substitution to depress the
# second example in exercise 1.6.
y=var('y')                  # prepare symbolic variable
p=x^3 + 3*x^2 + x + 3
q=p.substitute(x == y-1)
q                           # show output
```

Out[67]: $(y-1)^3 + 3\,(y-1)^2 + y + 2$

In [68]:
```
expand(q)                   # to multiply q out
```

Out[68]: $y^3 - 2\,y + 4$

In [69]:
```
redcubic(-2,4)              # now finds solution for y
```

Out[69]: $[1.00 - 1.00i, -2.00 + 0.0000103i, 1.00 + 1.00i]$

In [70]:
```
p.substitute(x == -3)       # with x=y-1 we can verify the root x=-3
```

Out[70]: 0

In [71]:
```
# We wish to use Sage to calculate the formula for depressing
# the general cubic polynomial.
l, m, n=var('l m n')                # prepare coefficients
p=x^3 + l*x^2 + m*x + n             # general cubic polynomial
expand(p.substitute(x == y-l/3))    # general reduction step
```

Out[71]: $\frac{2}{27}\,l^3 - \frac{1}{3}\,l^2 y + y^3 - \frac{1}{3}\,lm + my + n$

In [72]:
```
# We can also use the built-in command "solve" for quadratic,
# cubic, quartic, and other equations, but our focus remains
# practicing the mathematical ideas rather than learning all
# Sage commands. We note that the general solution (before
# depressing the equation) is not pretty and doesn't really fit).
solve(p, x)
```

Out[72]:

$$\left[x = -\frac{\left(l^2 - 3\,m\right)\left(-i\sqrt{3}+1\right)}{18\left(-\frac{1}{27}\,l^3 + \frac{1}{6}\,lm - \frac{1}{2}\,n + \frac{1}{6}\sqrt{-\frac{1}{3}\,l^2m^2 + \frac{4}{3}\,m^3 + \frac{2}{3}\left(2\,l^3 - 9\,lm\right)n + 9\,n^2}\right)^{\frac{1}{3}}} - \frac{1}{2}\left(-\frac{1}{27}\,l^3 + \frac{1}{6}\,lm - \frac{1}{2}\,n + \frac{1}{6}\sqrt{-\frac{1}{3}\,l^2m^2 + \frac{4}{3}\,m^3 + \frac{2}{3}\left(2\,l^3 - 9\,lm\right)n + 9\,n^2}\right)^{\frac{1}{3}}\left(i\right. \right.$$

$$-\frac{1}{3}\,l, x = -\frac{\left(l^2 - 3\,m\right)\left(i\sqrt{3}+1\right)}{18\left(-\frac{1}{27}\,l^3 + \frac{1}{6}\,lm - \frac{1}{2}\,n + \frac{1}{6}\sqrt{-\frac{1}{3}\,l^2m^2 + \frac{4}{3}\,m^3 + \frac{2}{3}\left(2\,l^3 - 9\,lm\right)n + 9\,n^2}\right)^{\frac{1}{3}}} - \frac{1}{2}\left(-\frac{1}{27}\,l^3 + \frac{1}{6}\,lm - \frac{1}{2}\,n + \frac{1}{6}\sqrt{-\frac{1}{3}\,l^2m^2 + \frac{4}{3}\,m^3 + \frac{2}{3}\left(2\,l^3 - 9\,lm\right)n + 9\,n^2}\right)^{\frac{1}{3}}\left(-\right.$$

$$\left. \left. -\frac{1}{3}\,l, x = -\frac{1}{3}\,l + \frac{l^2 - 3\,m}{9\left(-\frac{1}{27}\,l^3 + \frac{1}{6}\,lm - \frac{1}{2}\,n + \frac{1}{6}\sqrt{-\frac{1}{3}\,l^2m^2 + \frac{4}{3}\,m^3 + \frac{2}{3}\left(2\,l^3 - 9\,lm\right)n + 9\,n^2}\right)^{\frac{1}{3}}} + \left(-\frac{1}{27}\,l^3 + \frac{1}{6}\,lm - \frac{1}{2}\,n + \frac{1}{6}\sqrt{-\frac{1}{3}\,l^2m^2 + \frac{4}{3}\,m^3 + \frac{2}{3}\left(2\,l^3 - 9\,lm\right)n + 9\,n^2}\right)^{\frac{1}{3}} \right] \right.$$

In [73]:
```
def root(w=1, n=2):    # defines default values for arguments
  if w == 0:
     return [0 for k in range(n)]
  else:
     gam = angle(w)
     zlen = absval(w) ^ (1/n)
     return [(zlen * cis(gam/n+2*pi*k/n)).n(digits=3)
             for k in range(n)]
```

In [74]:
```
root(8, 3)
```

Out[74]: $[2.00, -1.00 + 1.73i, -1.00 - 1.73i]$

In [75]:
```
root(n=4)
```

Out[75]: $[1.00, 1.00i, -1.00, -1.00i]$

In [76]:
```
# The following gives an example for a while loop.
def firstpowertwo(n):
    k = 1
    while k <= n:        # while loop runs until condition fails
          k = 2*k
    print("The first power of 2 larger than "
          +str(n)+" is "+str(k)+".")
```

In [77]:
```
firstpowertwo(64)
```

```
The first power of 2 larger than 64 is 128.
```

IV. Recursion

In [78]:
```
# Just as induction is a powerful tool in mathematics, so is
# recursion in computer science. Here is an example.
def factorial(n):
    if n == 0:
        return 1                    # 0!=1
    else:
        return n * factorial(n-1)   # n!=n (n-1)!
```

In [79]:
```
factorial(6)
```

Out[79]: 720

In [80]:
```
reset("factorial")
```

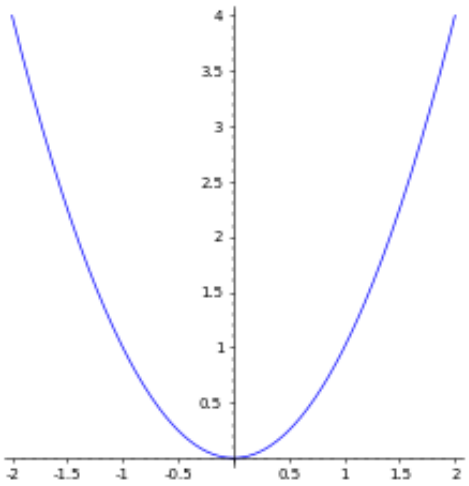
In [81]:
```
factorial(0.5)  # the build-in definition is more powerful
```

Out[81]: 0.886226925452758

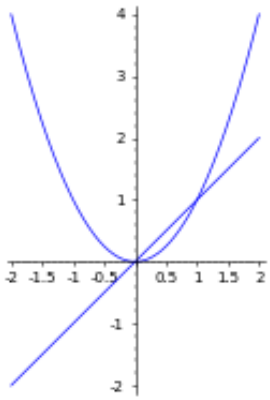
V. Plotting

In [82]:
```
# We use "aspect_ratio=1" to avoid a stretched picture and the
# dpi-command to change the dots-per-inch so that the graph fits.
plot(x^2, (x, -2, 2), aspect_ratio=1, dpi=50)
```

Out[82]:



In [83]:
```
# A Sage plot is also an object that can be stored under a variable
# and used for other purposes.
parabola = plot(x^2, (x, -2, 2))
myline = plot(x, (x, -2, 2))
(parabola + myline).show(aspect_ratio=1,dpi=50)
```



In [84]:
```
type(x)    # pre-defined variable
```

Out[84]:
```
<class 'sage.symbolic.expression.Expression'>
```

In [85]:
```
type(u)    # not yet defined
```

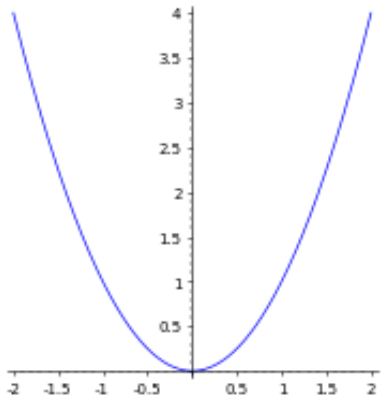
```
---------------------------------------------------------------------
---------
NameError                                 Traceback (most recent c
all last)
<ipython-input-85-80b41ec90327> in <module>()
----> 1 type(u)    # not yet defined

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

In [86]:
```
u = var("u") # initiates u as a variable
type(u)
```

Out[86]:
```
<class 'sage.symbolic.expression.Expression'>
```

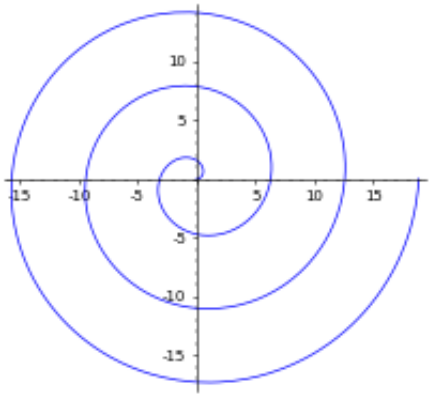
In [87]:
```
plot(u^2, (u, -2, 2), aspect_ratio=1, dpi=50) # using u
```

Out[87]:

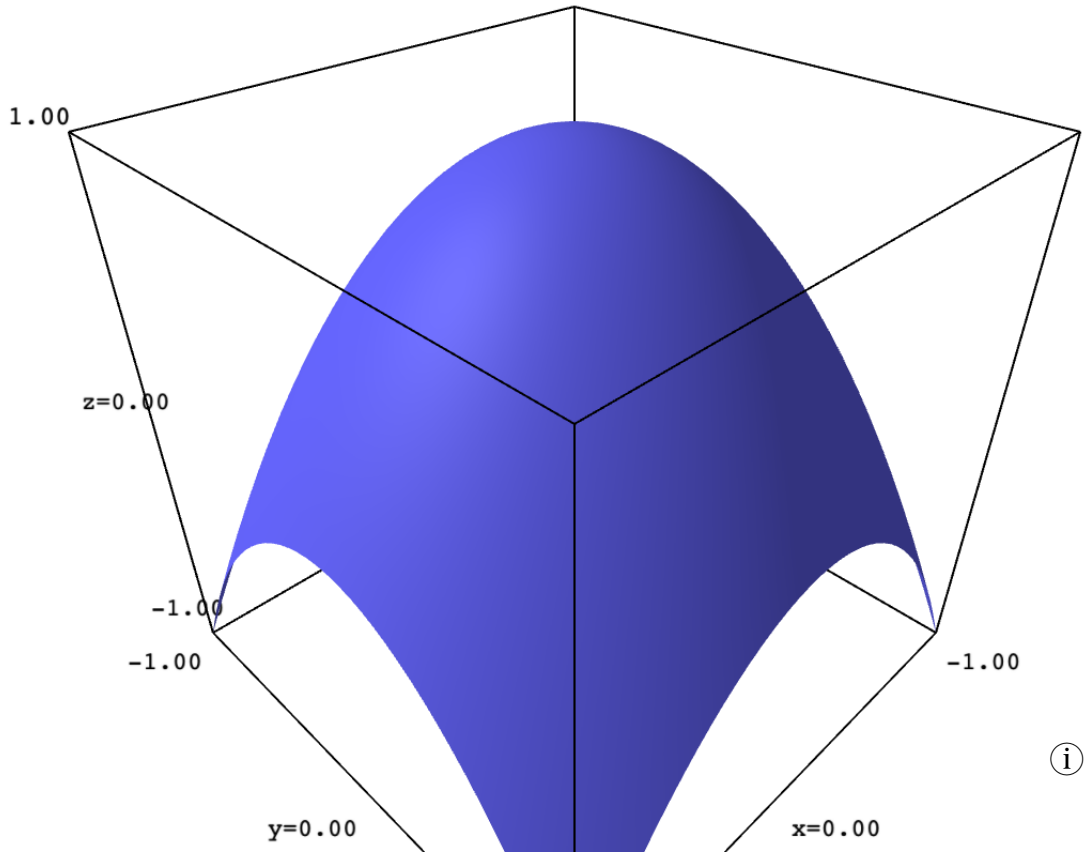


In [88]:
```
t = var("t") # parametric plot allows for more general curves
parametric_plot((t*cos(t), t*sin(t)), (t, 0, 6*pi),
                    aspect_ratio=1, dpi=50)
```

Out[88]:



In [89]:
```
x, y = var("x y")
tent = plot3d(1 - x^2 - y^2, (x, -1, 1), (y, -1, 1)) #3D graph
tent.save("tent.bmp")
tent.show() # shows the surface which, depending on software,
            # can be rotated (not in a pdf)
```

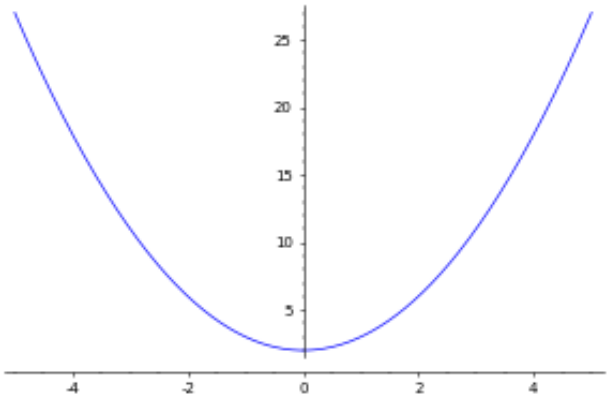


VI. Projects

In [90]:
```
def parabola(a, b, c):
    return plot(a*x^2 + b*x + c, (x, -5, 5), dpi=50)
```

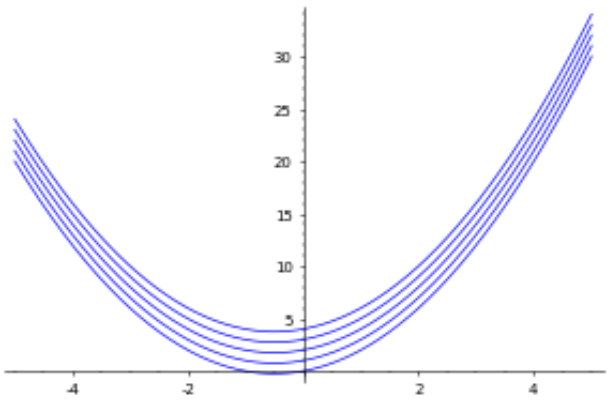
In [91]:
```
parabola(1, 0, 2) # aspect_ratio=1 would make the graph very high
```

Out[91]:



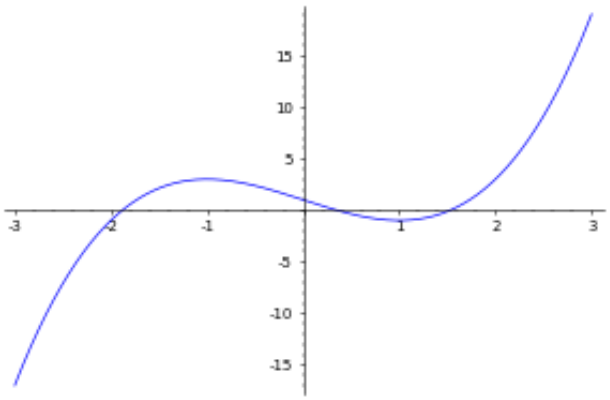
In [92]:
```
sum([parabola(1, 1, k) for k in range(5)])
```

Out[92]:



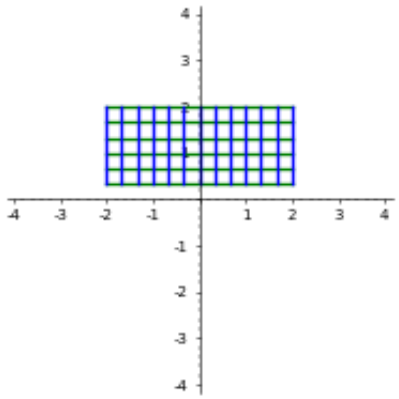
In [93]:
```
def cubic(p, q):
    return plot(x^3 + p*x + q, (x, -3, 3))
```

In [94]:
```
cubic(-3, 1).show(dpi=50)
```



In [95]:
```
def grid(n):
    hor_lines=sum([line([[-2, 2*k/n], [2, 2*k/n]], color="green",
                        thickness=2)
                   for k in range(1,n+1)])
    ver_lines=sum([line([(2*k/n, 2/n), (2*k/n, 2)], color="blue",
                        thickness=2)
                   for k in range(-n,n+1)])
    return (hor_lines + ver_lines)
    # a grid in the complex plane
```

In [96]:
```
grid(6).show(aspect_ratio=1, dpi=50,
             xmin=-4, xmax=4,
             ymin=-4, ymax=4)
```



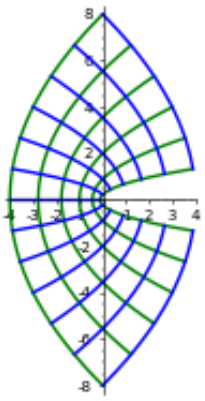
In [97]:
```
# The following simple command converts complex numbers into
# a Cartesian tuple.
def cart(z):
    return (z.real(), z.imag())
```

In [98]:
```
cart(i)
```

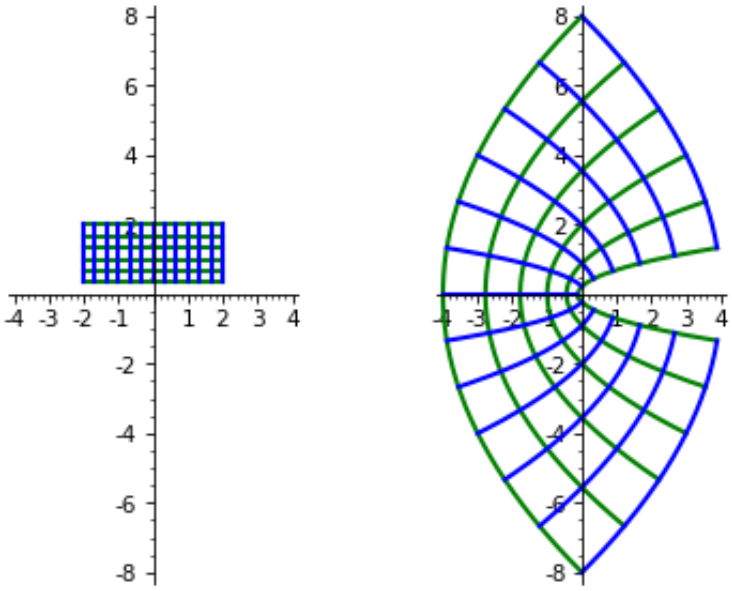
Out[98]: $(0, 1)$

In [99]:
```
# The following draws the image of the grid under the squaring map.
t = var("t")
def gridsquare(n):
    hor = sum([parametric_plot(cart((t+i*2*k/n)^2), (t,-2,2),
                               color="green",thickness=2)
               for k in range(1,n+1)])
    ver = sum([parametric_plot(cart((2*k/n+i*t)^2), (t,2/n,2),
                               color="blue",thickness=2)
               for k in range(-n,n+1)])
    return hor+ver
```

In [100]:
```
gridsquare(6).show(aspect_ratio=1, dpi=50)
```



In [101]:
```
# The following picture indicates how the squaring map on the
# complex plane looks like.
graphics_array((grid(6), gridsquare(6))).\
    show(aspect_ratio=1, dpi=75, xmin=-4, xmax=4, ymin=-8, ymax=8)
```



In [102]:
```
t=var("t")
frames=[parametric_plot((cos(t), sin(t)), (t, 0, b))
            for b in srange(0.001, 2*pi, pi/60)]\
       +[circle((0, 0), 1) for n in range(10)] # circle stays
cir_mov = animate(frames, xmin=-1, ymin=-1, xmax=1, ymax=1,
                  aspect_ratio=1)
# This creates an animation that draws a circle and saved as movie.
cir_mov.save(filename="circle.mp4", use_ffmpeg=True)
```

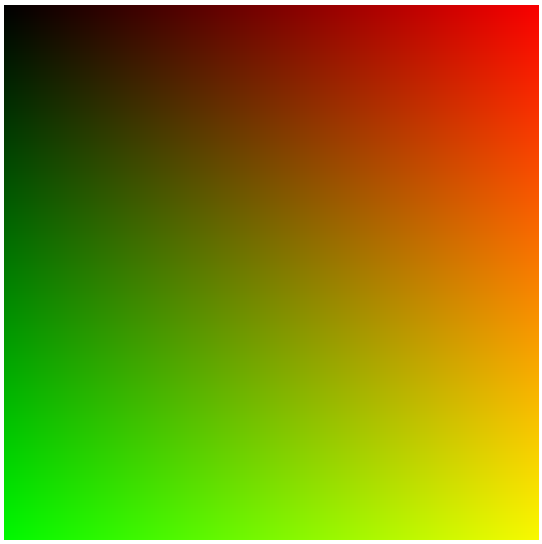
In [103]:
```
# The following creates a circle rolling along a circle,
# which creates a cardiode if we trace one of the points.
fixcircle=circle((0,0),1)
s=var("s")
roll=[fixcircle
      +circle((2*cos(t), 2*sin(t)), 1)
      +point((2*cos(t) - cos(2*t), 2*sin(t) - sin(2*t)),
             size=16, color="green")
      +parametric_plot((2*cos(s) - cos(2*s), 2*sin(s) - sin(2*s)),
                       (s, 0, t), color="green")
      for t in sxrange(0.01, 2*pi, 0.02)]
roll_ani = animate(roll, xmin=-4, xmax=4, ymin=-4, ymax=4)
roll_ani.save(filename="rolling.mp4", use_ffmpeg=True)
```

In [104]:
```
# The following animation shows that the winding number starts
# with 0 but goes up to three, where each jump of the
# winding number corresponds to a root of p(z).
cis(phi) = cos(phi) + i*sin(phi)
p(z) = z^3 + i*z^2 + z - 1
t = var("t")
frames = [parametric_plot(cart(p(r*cis(t))), (t, 0, 2*pi))
          for r in sxrange(0, 2.5, 0.01)]\
         +[parametric_plot(cart(p(2.5*cis(t))), (t, 0, 2*pi))
           for n in range(0, 40)]
fund = animate(frames, xmin=-20, xmax=20, ymin=-20, ymax=20)
fund.save(filename="fundamental.mp4", use_ffmpeg=True)
```

In [105]:
```
def pythagoras(a): # outputs the two pictures for Pythagoras
    triang=polygon([(0,0), (1,0), (1,1), (0,1)],color='grey')\
           +polygon([(a,0), (1,a), (1-a,1), (0,1-a)],color='red')
    binom=polygon([(0,0), (1,0), (1,1), (0,1)],color='grey')\
          +polygon([(0,0), (a,0), (a,a), (0,a)],color='red')\
          +polygon([(a,a), (1,a), (1,1), (a,1)],color='red')
    return graphics_array((binom, triang)) # side by side
```

In [106]:
```
# The following animates a proof of Pythogoras theorem.
frames=[pythagoras(a) for a in sxrange(0, 1, 0.01)]\
       +[pythagoras(1-a) for a in sxrange(0, 1, 0.01)]
pythmovie = animate(frames)
pythmovie.save(filename="pythagoras.mp4", use_ffmpeg=True)
```

In [107]:
```
# We indicate briefly how to work with bitmaps:
from sage.repl.image import Image # cryptic: defines command Image
resolution = 250
# The following line now creates a bitmap in RGB-format.
justcolors = Image("RGB", (resolution, resolution))

for u in range(resolution):
    for v in range(resolution):
        justcolors.pixels()[u, v] = (u, v, 0) # changes pixel color
justcolors.show()
```



In [108]:
```
for u in range(resolution):
    for v in range(resolution):
        justcolors.pixels()[u, v]=(u, v, 255 - (u+v)/2)
justcolors.show()
```

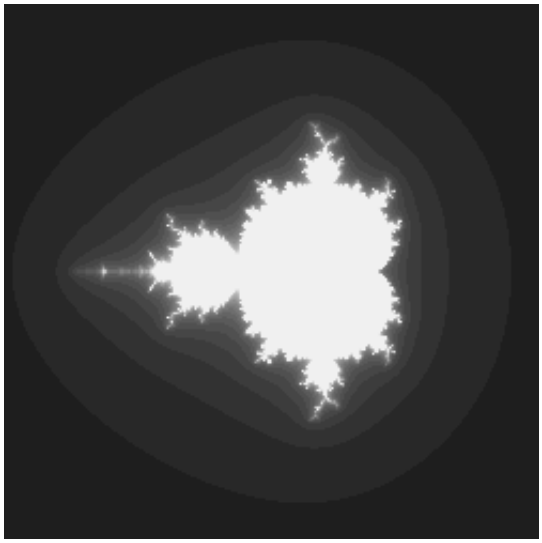
In [109]:
```
# The following is the iteration for Mandelbrot and Julia sets.
max=25
def iterate(c, z=0.0):
    for n in range(max):
        z = z^2 + c
        if abs(z) > 100:
            break
    return n
```

In [110]:
```
# How often do we have to apply z^2+.4 to get from 0 to over 100?
iterate(.4)
```

Out[110]: 8

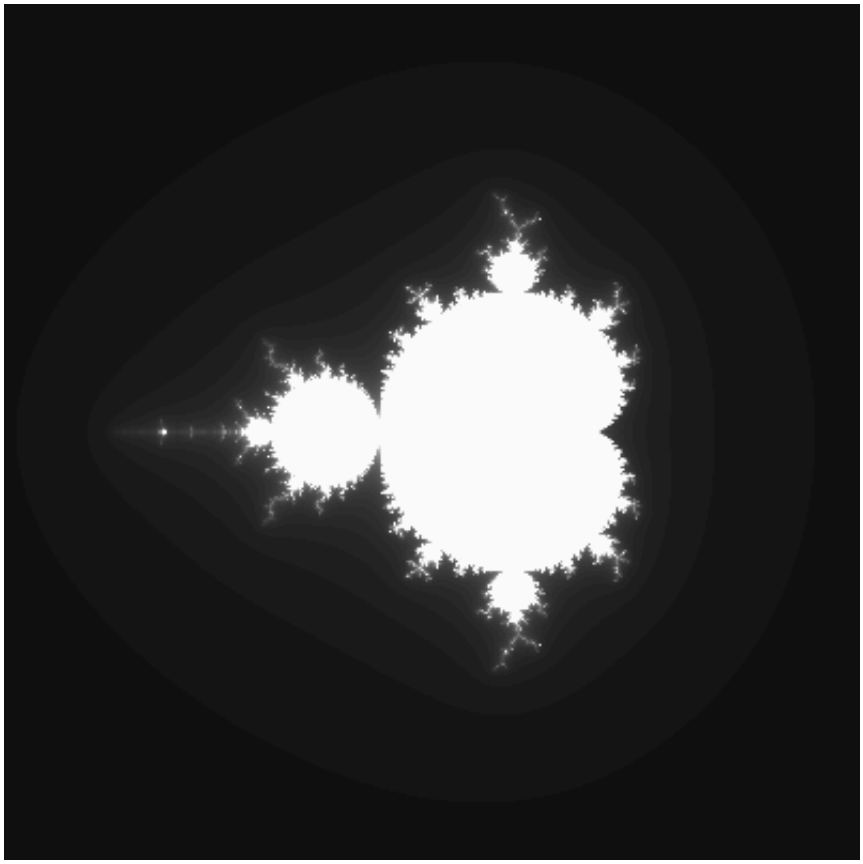
In [111]:
```
MBset=Image("RGB",(resolution,resolution))
for u in range(resolution):
    for v in range(resolution/2):
        cx = -2.5 + u*4/resolution
        cy = 2 - v*4/resolution
        c = cx + i*cy
        it = iterate(c.n())
        MBset.pixels()[u, v]=(it*10, it*10, it*10)
        MBset.pixels()[u, int(resolution-1-v)]=(it*10,it*10,it*10)
```

In [112]:
```
MBset.show()
```



In [113]:
```
max=51          # better quality, much longer calculation
resolution=400 # bigger picture, much longer calculation
MBset=Image("RGB",(resolution,resolution))
for u in range(resolution):
    for v in range(resolution/2):
        cx = -2.5 + u*4/resolution
        cy = 2 - v*4/resolution
        c = cx + i*cy
        it = iterate(c.n())
        MBset.pixels()[u, v] = (it*5, it*5, it*5)
        MBset.pixels()[u, int(resolution-1-v)] = (it*5, it*5, it*5)
```

In [114]:
```
MBset.show()
```



In [115]:
```
@interact     #allows us to change cx and cy using sliders
def cross(cx=(-2.4,1.4,0.05), cy=(-1.9,1.9,0.05)):
    MBexperiment=deepcopy(MBset)
    u=int((cx+2.5) / 4 * resolution)
    v=int((2-cy) / 4 * resolution)
    MBexperiment.pixels()[u, v]=(255, 255, 255)
    MBexperiment.pixels()[int(u+1), int(v+1)]=(255, 255, 255)
    MBexperiment.pixels()[int(u-1), int(v+1)]=(255, 255, 255)
    MBexperiment.pixels()[int(u-1), int(v-1)]=(255, 255, 255)
    MBexperiment.pixels()[int(u+1), int(v-1)]=(255, 255, 255)
    MBexperiment.pixels()[int(u+1), v]=(40, 40, 255)
    MBexperiment.pixels()[int(u+2), v]=(40, 40, 255)
    MBexperiment.pixels()[int(u-1), v]=(40, 40, 255)
    MBexperiment.pixels()[int(u-2), v]=(40, 40, 255)
    MBexperiment.pixels()[u, int(v+1)]=(40, 40, 255)
    MBexperiment.pixels()[u, int(v+2)]=(40, 40, 255)
    MBexperiment.pixels()[u, int(v-1)]=(40, 40, 255)
    MBexperiment.pixels()[u, int(v-2)]=(40, 40, 255)
    MBexperiment.show()
```

In [116]:
```
cx0 = -0.35   # x of top left corner of window, found above
cy0 = 1       # y of top left corner of window, found above
max = 51
MBzoom=Image("RGB", (resolution, resolution))
for u in range(resolution):
    for v in range(resolution):
        cx = cx0 + u/(2*resolution)
        cy = cy0 - v/(2*resolution)
        c = cx + i*cy
        it = iterate(c.n())
        MBzoom.pixels()[u, v]=(it*5, it*5, it*5)
```

In [117]:
```
MBzoom.show()
```

In [118]:
```
# The following draws a circle on a bitmap.
def draw_circ(pic, cx, cy, r):
    for phi in sxrange(0, 2*pi, 0.01):
        x = cx + r*cos(phi)
        y = cy + r*sin(phi)
        u = (x+2.5) / 4 * resolution
        v = (2-y)/ 4 * resolution
        if u > 0 and u < resolution and v > 0 and v < resolution:
            pic.pixels()[int(u), int(v)]=(100, 100, 255)
```

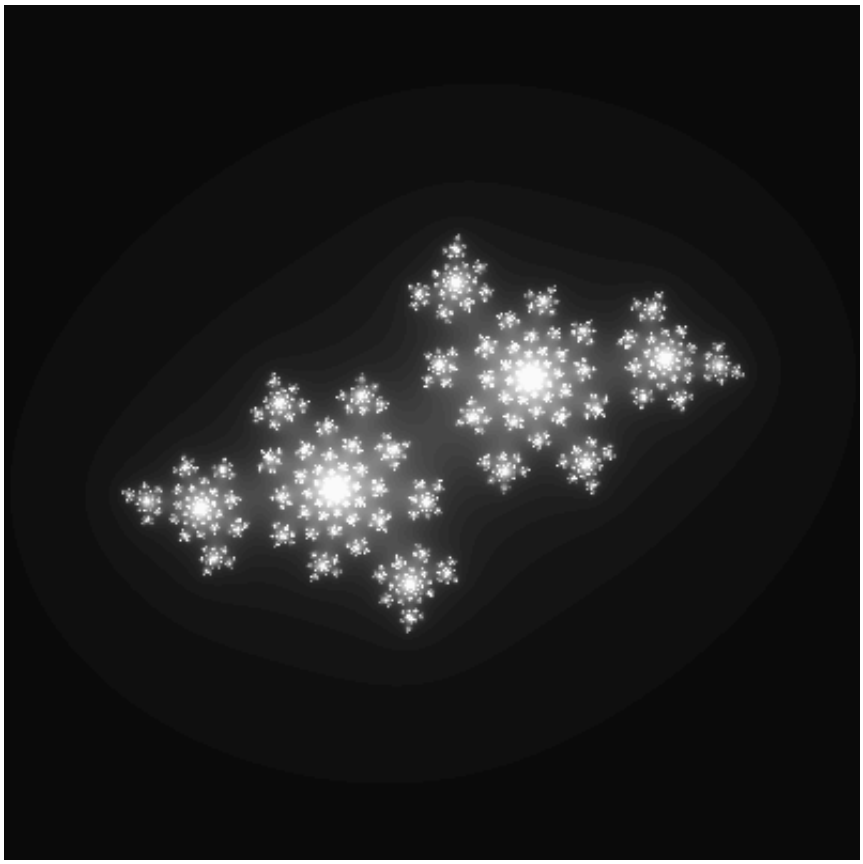
In [119]:
```
@interact
def bulb2(cx=(-2.4, 1.4, 0.05),r=(0.1, 1.5, 0.05)):
    MBexperiment = deepcopy(MBset)
    draw_circ(MBexperiment, cx, 0, r)
    MBexperiment.show()   # second bulb has a circle as boundary
```

In [120]:
```
# The following draws a cardiode on a bitmap.
def draw_card(pic, cx, cy, r):
    for phi in sxrange(0, 2*pi , 0.01):
        x = cx + r*(2*cos(phi)-cos(2*phi))
        y = cy + r*(2*sin(phi)-sin(2*phi))
        u = (x+2.5) / 4 * resolution
        v = (2-y) / 4 * resolution
        if u>0 and u<resolution and v>0 and v<resolution:
            pic.pixels()[int(u), int(v)]=(100, 100, 255)
```

In [121]:
```
@interact
def bulb1(cx=(-2.4, 1.4, 0.05),r=(0.1, 1.5, 0.05)):
    MBexperiment=deepcopy(MBset)
    draw_card(MBexperiment, cx, 0, r)
    MBexperiment.show() # first bulb has a cardiode as boundary
```

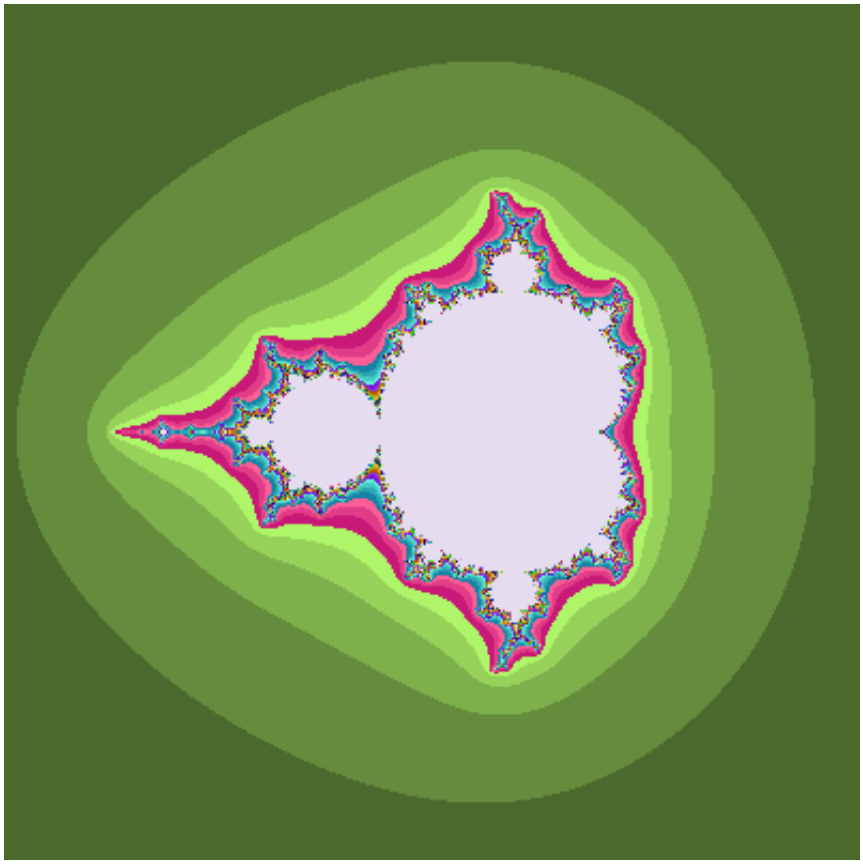
In [122]:
```
cx = -0.6
cy = -0.5
c = cx + i*cy
# For the Julia set we fix the parameter c and change the value
# of z for the recursion depending on the current pixel.
Julia = Image("RGB",(resolution,resolution))
for u in range(resolution):
    for v in range(resolution):
        zx = -2 + u*4/resolution
        zy = 2 - v*4/resolution
        z = zx + i*zy
        it = iterate(c, z.n())
        Julia.pixels()[u, v]=(it*5, it*5, it*5)
```

In [123]:
```
Julia.show()
```



In [124]:
```
# The following allows us to experiment with the colors without
# having to repeat all the calculations done already.
def color(bitmap):
    new = Image("RGB", (resolution, resolution))
    for u in range(resolution):
        for v in range(resolution):
            old = bitmap.pixels()[u, v][0]
            new.pixels()[u, v] = (5*old%255, 7*old%255, 3*old%255)
            # % calculates the remainder for integer division
    new.show()
```

In [125]:
```
color(MBset)
```



In [ ]: