Chapter 1. Mathematics and Python

1.1. Propositional logic

Program: table1.py

In [1]:
```
for P in [True, False]:
    print(P, not P)
```

```
True False
False True
```

Program: table2.py

In [1]:
```
for P in [True, False]:
    for Q in [True, False]:
        print(P, Q, P and Q, P or Q, P <= Q , P == Q)
```

```
True True True True True True
True False False True False False
False True False True True False
False False False False True True
```

Program: demorgan1.py

In [1]:
```
for P in [False, True]:
    for Q in [False, True]:
        print(P, Q, not P, not Q, P and Q,
              not(P and Q), (not P) or (not Q))
```

```
False False True True False True True
False True True False False True True
True False False True False True True
True True False False True False False
```

Table 1.1.

logic_table.py

P	$\neg P$
true	false
false	true

P	Q	$P \wedge Q$	$P \vee Q$	$P \rightarrow Q$	$P \leftrightarrow Q$
true	true	true	true	true	true
true	false	false	true	false	false
false	true	false	true	true	false
false	false	false	false	true	true

1.2. Numbers

Program: eqn1.py

In [1]:
```
from sympy import solve, I
from sympy.abc import x, y

ans1 = solve([x + 2 * y - 1, 4 * x + 5 * y - 2], [x, y])
print(ans1)
ans2 = solve(x ** 2 + x + 1, x)
print(ans2)
```

```
{x: -1/3, y: 2/3}
[-1/2 - sqrt(3)*I/2, -1/2 + sqrt(3)*I/2]
```

Untitled.ipynb

In [1]:
```
x = 1 + 2j
x.real, x.imag, abs(x), x.conjugate()
```

Out[1]:
```
(1.0, 2.0, 2.23606797749979, (1-2j))
```

In [2]:
```
y = 3 + 4j
x + y, x * y, x / y
```

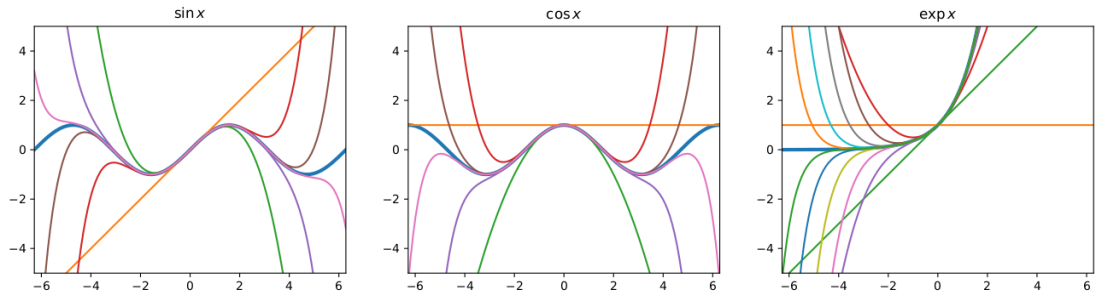
Out[2]:
```
((4+6j), (-5+10j), (0.44+0.08j))
```

In [3]:
```
2.718281828459045 ** 3.141592653589793j
```

Out[3]:
```
(-1+1.2246467991473532e-16j)
```

Fig. 1.1.

taylor.py



1.3. Sets

Untitled.ipynb

In [1]:
```
A = {2, 3, 5, 7}; A
```

Out[1]:
```
{2, 3, 5, 7}
```

In [2]:
```
B = set([6, 9, 3]); B
```

Out[2]:
```
{3, 6, 9}
```

In [3]:
```
C = {3, 6, 9, 6, 3}; C
```

Out[3]:
```
{3, 6, 9}
```

In [4]:
```
set()
```

Out[4]:
```
set()
```

In [5]:
```
C == {3, 6, 9}
```

Out[5]:
```
True
```

In [6]:
```
{x for x in range(2, 10) if all([x%n for n in range(2, x)])}2 in A
```

Out[6]:
```
True
```

In [7]:
```
2 in B
```

Out[7]:
```
False
```

In [8]:
```
A | B
```

Out[8]:
```
{2, 3, 5, 6, 7, 9}
```

In [9]:
```
A & B
```

Out[9]:
```
{3}
```

In [10]:
```
(A & B).issubset(A)
```

Out[10]:
```
True
```

In [11]:
```
A.issubset(A | B)
```

Out[11]:
```
True
```

In [12]:
```
A | B == A.union(B)
```

Out[12]:
```
True
```

In [13]:
```
A & B == A.intersection(B)
```

Out[13]:
```
True
```

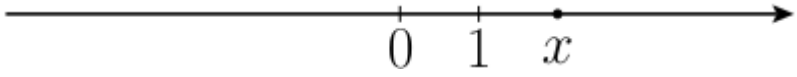
Untitled1.ipynb

In [1]:
```
{x for x in range(2, 10) if all([x%n for n in range(2, x)])}
```

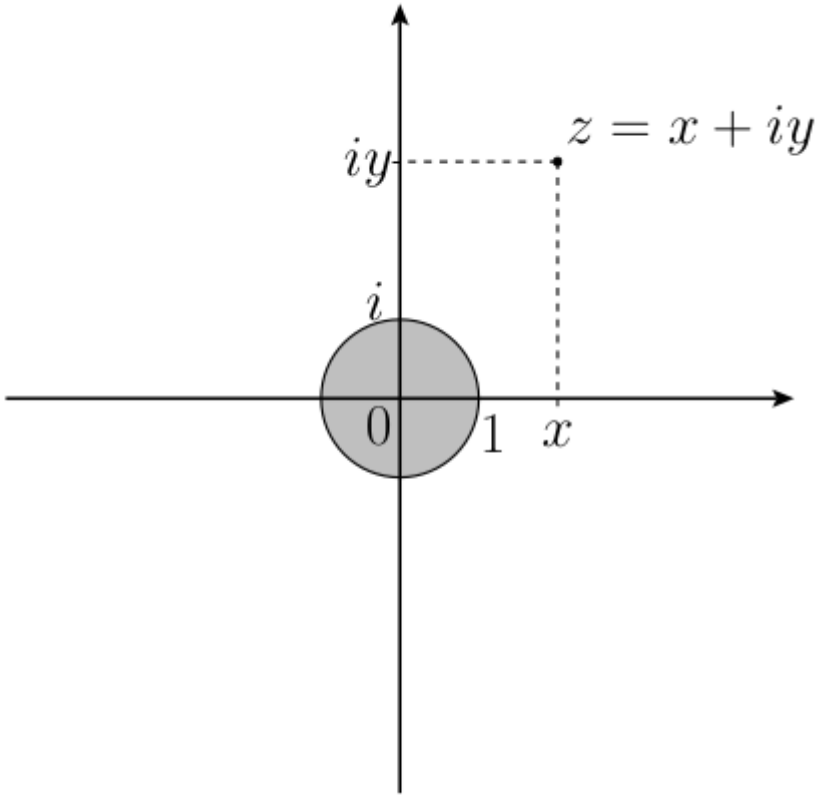
Out[1]:
```
{2, 3, 5, 7}
```

Fig. 1.2.

real.py



complex.py



1.4. Ordered pairs and tuples

Untitled.ipynb

In [1]:
```
x = (1, 2); x
```

Out[1]:
```
(1, 2)
```

In [2]:
```
y = (2, 1); y
```

Out[2]:
```
(2, 1)
```

In [3]:
```
z = (1, 2, 1); z
```

Out[3]:
```
(1, 2, 1)
```

In [4]:
```
x == y, x == z, y == z
```

Out[4]:
```
(False, False, False)
```

In [5]:
```
set(x) == set(y), set(x) == set(z), set(y) == set(z)
```

Out[5]: (True, True, True)

In [6]:
```
set(x), set(y), set(z)
```

Out[6]: ({1, 2}, {1, 2}, {1, 2})

Untitled1.ipynb

In [1]:
```
a = (2, 3, 5, 7); a
```

Out[1]: (2, 3, 5, 7)

In [2]:
```
a[0], a[1], a[2], a[3]
```

Out[2]: (2, 3, 5, 7)

In [3]:
```
a[-1], a[-2], a[-3], a[-4]
```

Out[3]: (7, 5, 3, 2)

In [4]:
```
x = (1,); x
```

Out[4]: (1,)

In [5]:
```
x[0]
```

Out[5]: 1

1.5. Mappings and functions

Untitled.ipynb

In [ ]:
```
def f(x):
    return x**2 - 1
```

In [ ]:
```
def g(x):
    if x < 0:
        return 0
    else:
        return 1
```

In [ ]:
```
def g(x):
    return 0 if x < 0 else 1
```

In [ ]:
```
lambda x: x**2 - 1
```

In [ ]:
```
def f(x):
    if n == 0:
        return 1
    else:
        return n * f(n-1)
```

1.6. Classes and objects in Python

Untitled.ipynb

In [1]:
```
A = 'Hello Python!'; A
```

Out[1]:
```
'Hello Python!'
```

In [2]:
```
print(A)
```

```
Hello Python!
```

In [3]:
```
print(A[0], A[1], A[2], A[3])
```

```
H e l l
```

In [4]:
```
print(A[-1], A[-2], A[-3], A[-4])
```

```
! n o h
```

Untitled1.ipynb

In [1]:
```
B = ['Earth', 'Mars', 'Jupiter']; B
```

Out[1]:
```
['Earth', 'Mars', 'Jupiter']
```

In [2]:
```
print(B[0], B[1], B[2])
```

```
Earth Mars Jupiter
```

In [3]:
```
print(B[-1], B[-2], B[-3])
```

```
Jupiter Mars Earth
```

In [4]:
```
B.append('Saturn'); B
```

Out[4]:
```
['Earth', 'Mars', 'Jupiter', 'Saturn']
```

Untitled2.ipynb

In [1]:
```
C = {'Earth': '3rd', 'Mars': '4th', 'Jupiter': '5th'}; C
```

Out[1]:
```
{'Earth': '3rd', 'Mars': '4th', 'Jupiter': '5th'}
```

In [2]:
```
C['Earth']
```

Out[2]:
```
'3rd'
```

In [3]:
```
C['Saturn'] = '6th'; C
```

Out[3]:
```
{'Earth': '3rd', 'Mars': '4th', 'Jupiter': '5th', 'Saturn': '6th'}
```

Untitled3.ipynb

In [1]:
```
x, y, z = 1, 2, 3; x, y, z
```

Out[1]:
```
(1, 2, 3)
```

In [2]:
```
f'{x}+{y}+{z}'
```

Out[2]:
```
'1+2+3'
```

In [3]:
```
'{}+{}+{}'.format(x, y, z)
```

Out[3]:
```
'1+2+3'
```

In [4]:
```
'%s+%s+%s' % (x, y, z)
```

Out[4]:
```
'1+2+3'
```

In [5]:
```
A = [z, y, x]; A
```

Out[5]:
```
[3, 2, 1]
```

In [6]:
```
sorted(A)
```

Out[6]:
```
[1, 2, 3]
```

In [7]:
```
A
```

Out[7]:
```
[3, 2, 1]
```

In [8]:
```
A.sort()
A
```

Out[8]:
```
[1, 2, 3]
```

1.7. Lists, arrays and matrices

Untitled.ipynb

In [1]:
```
A = (1, 2, 3); A
```

Out[1]:
```
(1, 2, 3)
```

In [2]:
```
B = [1, 2, 3]; B
```

Out[2]:
```
[1, 2, 3]
```

In [3]:
```
A[0], B[0]
```

Out[3]: (1, 1)

In [4]:
```
B[0] = 0; B
```

Out[4]: [0, 2, 3]

In [5]:
```
A[0] = 0
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In [5], line 1
----> 1 A[0] = 0

TypeError: 'tuple' object does not support item assignment
```

Program: matrix.py

In [1]:
```
A = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
for i in range(3):
    for j in range(3):
        print(f'A[{i}][{j}]=={A[i][j]}', end=', ')
    print()
```

```
A[0][0]==1, A[0][1]==2, A[0][2]==3,
A[1][0]==4, A[1][1]==5, A[1][2]==6,
A[2][0]==7, A[2][1]==8, A[2][2]==9,
```

1.8. Preparation of image data

Binarization of image data with PIL and NumPy

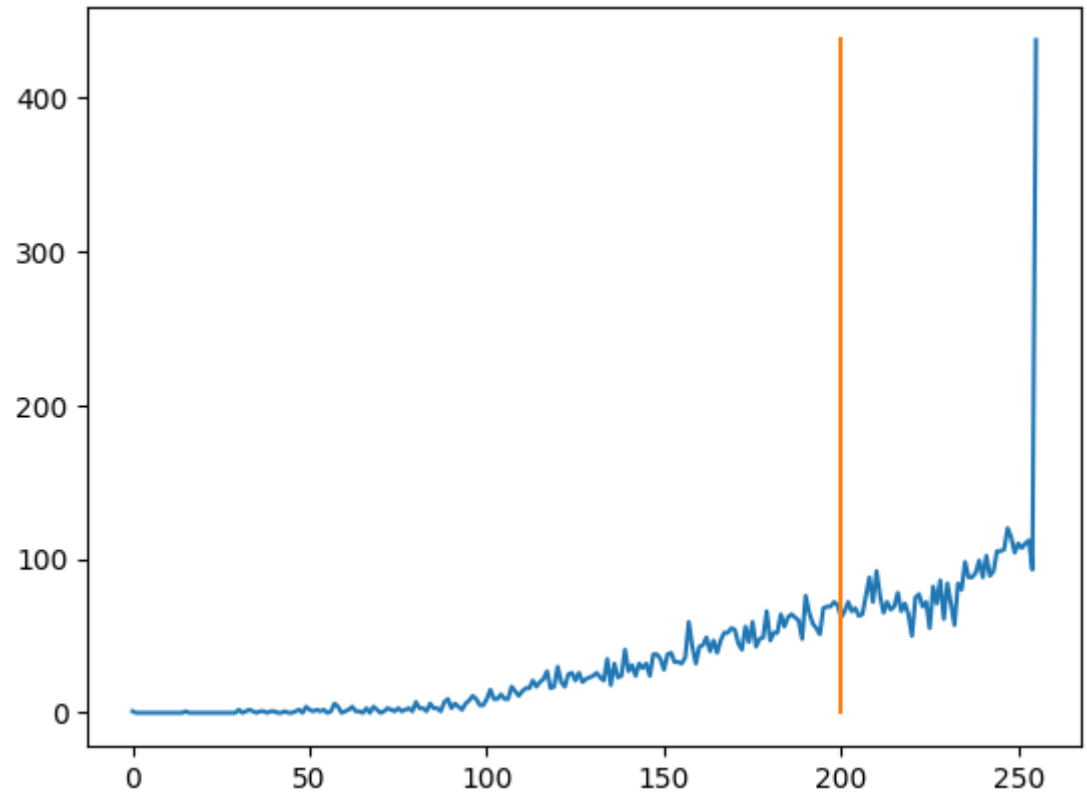
Program: graph1.py

In [1]:
```
from numpy import array, zeros
import PIL.Image as Img
import matplotlib.pyplot as plt

im = Img.open('mypict1.jpg')
A = array(im)
m, n = A.shape
avr = A.sum()/m/n
print((m, n), avr)

L = zeros(256)
for i in range(m):
    for j in range(n):
```

```
        L[A[i, j]] += 1

plt.plot(range(256), L)
plt.plot([avr, avr], [0, L.max()])
```

```
(100, 88) 199.83977272727273
```

Out[1]:
```
[<matplotlib.lines.Line2D at 0x7f5a074bb0>]
```



Program: mypict1.py

In [1]:
```
import PIL.Image as Img
from numpy import array

A = array(Img.open('1.8/mypict1.jpg'))
B = A < 200
m, n = B.shape
h = max(m, n)
x0, y0 = m / h, n / h

def f(i, j):
    return (y0 * (-1 + 2 * j / (n - 1)), x0 * (1 - 2 * i / (m - 1)))

P = [f(i, j) for i in range(m) for j in range(n) if B[i, j]]
with open('mypict1.txt', 'w') as fd:
    fd.write(repr(P))
```

In [2]:
```
B
```

```
Out[2]: array([[False, False, False, ..., False,  True, False],
               [False, False, False, ...,  True,  True, False],
               [False, False, False, ...,  True,  True, False],
               ...,
               [False, False, False, ..., False, False, False],
               [False, False, False, ..., False, False, False],
               [False, False, False, ..., False, False, False]])
```

Untitled.ipynb

```
In [1]: from numpy import *
```

```
In [2]: A = array([1, 2, 3]); A
```

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

```
In [3]: print(A)
```

```
[1 2 3]
```

```
In [4]: str(A)
```

```
Out[4]: '[1 2 3]'
```

```
In [5]: repr(A)
```

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

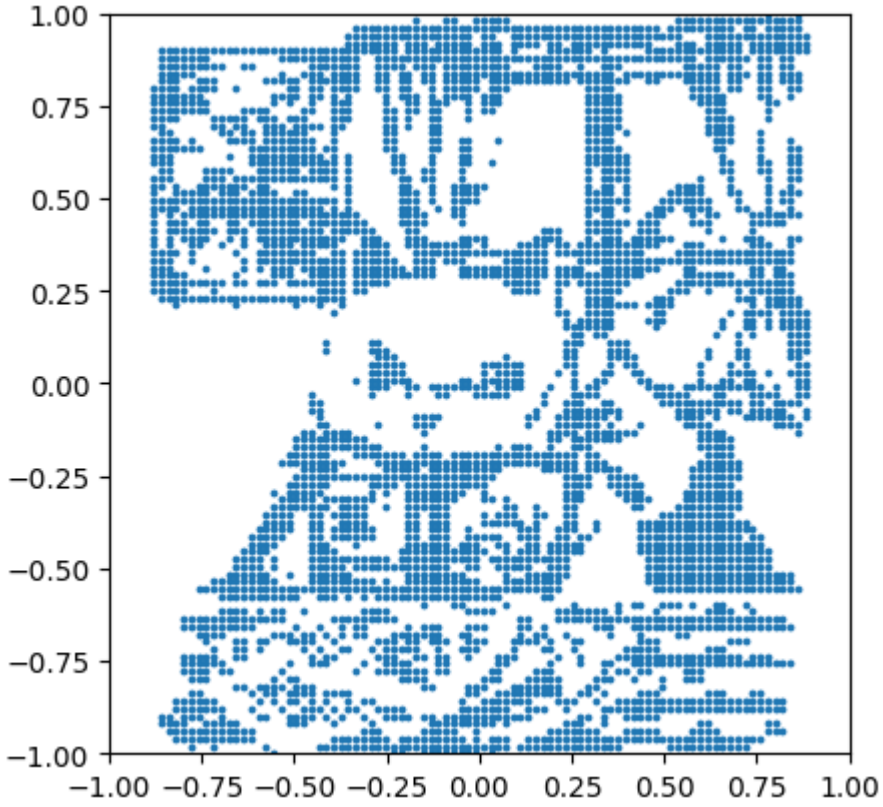
```
In [6]: eval('array([1, 2, 3])')
```

```
Out[6]: array([1, 2, 3])
```

Program: mypict2.py

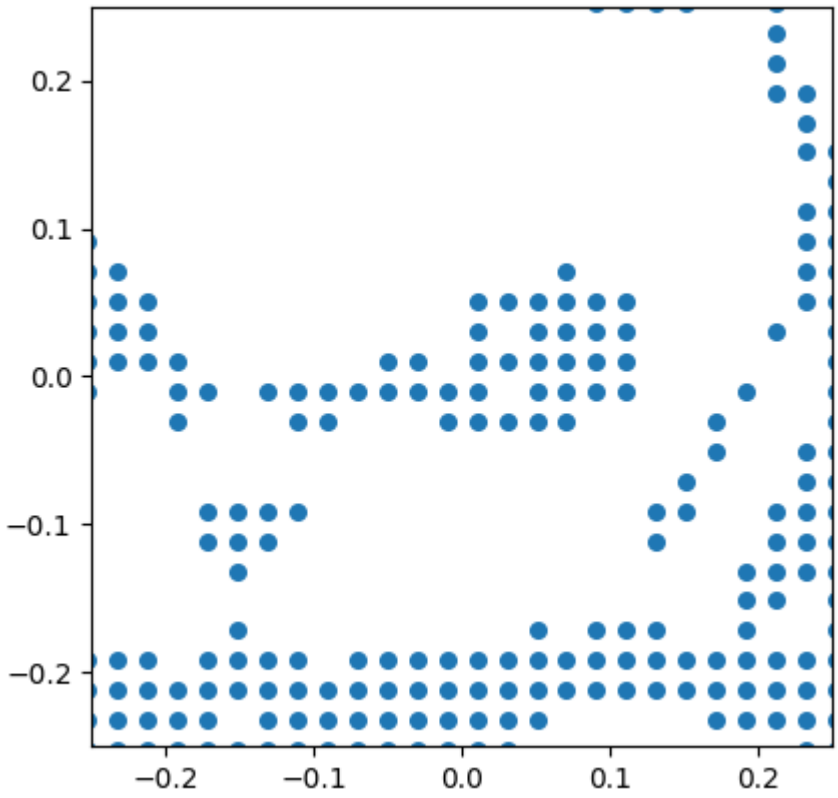
```
In [1]: import matplotlib.pyplot as plt

with open('mypict1.txt', 'r') as fd:
    P = eval(fd.read())
x, y = zip(*P)
plt.scatter(x, y, s=3)
plt.axis('scaled'), plt.xlim(-1, 1), plt.ylim(-1, 1)
```

```
Out[1]: ((-0.968, 0.968, -1.1, 1.1), (-1.0, 1.0), (-1.0, 1.0))
```



In [2]:
```
import matplotlib.pyplot as plt

with open('mypict1.txt', 'r') as fd:
    P = eval(fd.read())
x, y = zip(*P)
plt.scatter(x, y, s=30)
plt.axis('scaled'), plt.xlim(-0.25, 0.25), plt.ylim(-0.25, 0.25)
```

Out[2]: ((-0.968, 0.968, -1.1, 1.1), (-0.25, 0.25), (-0.25, 0.25))

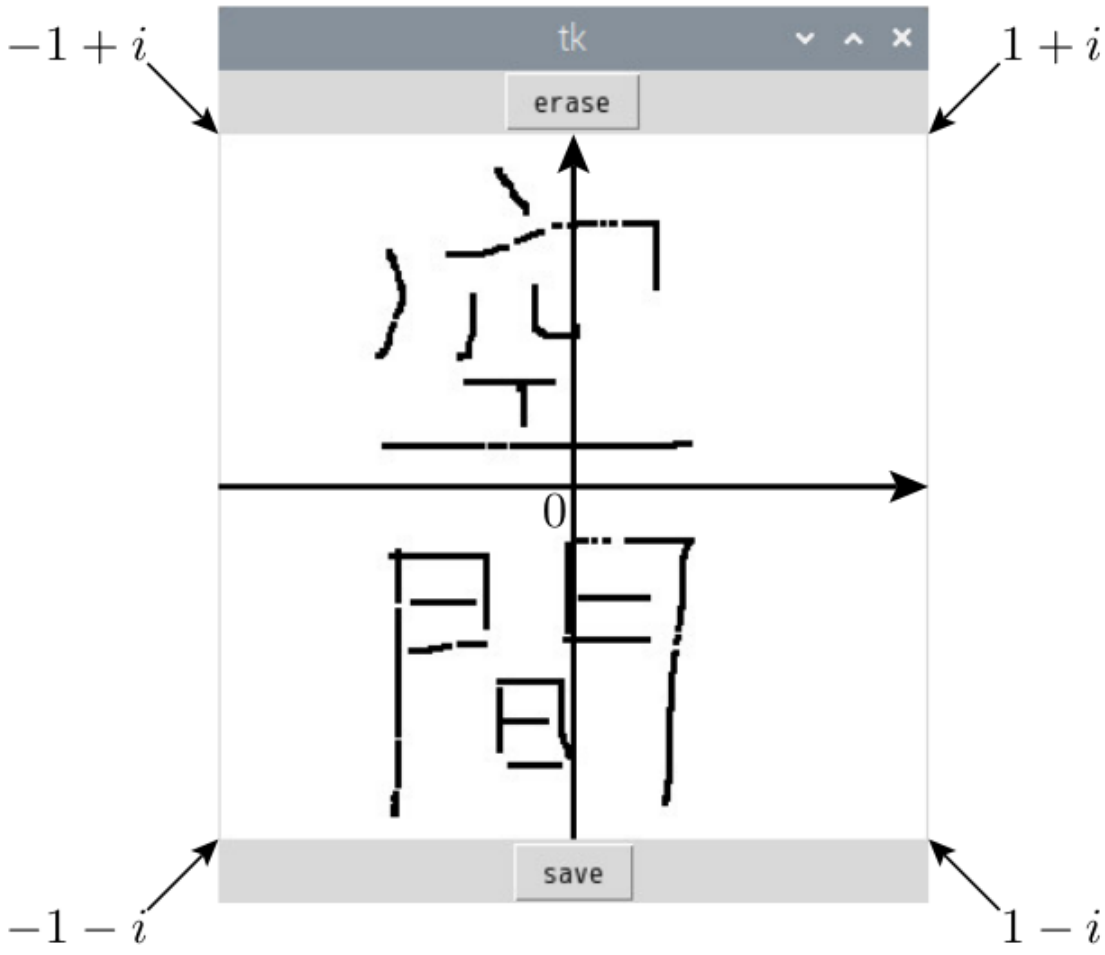


GUI for creating complex-valued data of handwritten characters

Program: tablet.py

In [1]:
```
from tkinter import Tk, Button, Canvas

def point(x, y):
    return C.create_oval(x - 1, y - 1, x + 1, y + 1, fill='black')

def PushButton(event):
    x, y = event.x, event.y
    Segs.append([((x, y), point(x, y))])

def DragMouse(event):
    x, y = event.x, event.y
    Segs[-1].append(((x, y), point(x, y)))

def Erase():
    if Segs != []:
        seg = Segs.pop()
        for p in seg:
            C.delete(p[1])

def Save():
    if Segs != []:
        L = []
        for seg in Segs:
            for (x, y), _ in seg:
                L.append((x - 160) / 160 + 1j * (160 - y) / 160)
        with open(filename, 'w') as fd:
            fd.write(repr(L))
        print('saved!')

filename = 'tablet.txt'
Segs = []
tk = Tk()
Button(tk, text="erase", command=Erase).pack()
C = Canvas(tk, width=320, height=320, bg='white')
C.pack()
C.bind("<Button-1>", PushButton)
C.bind("<B1-Motion>", DragMouse)
Button(tk, text="save", command=Save).pack()
tk.mainloop()
```

tablet_pyx.py

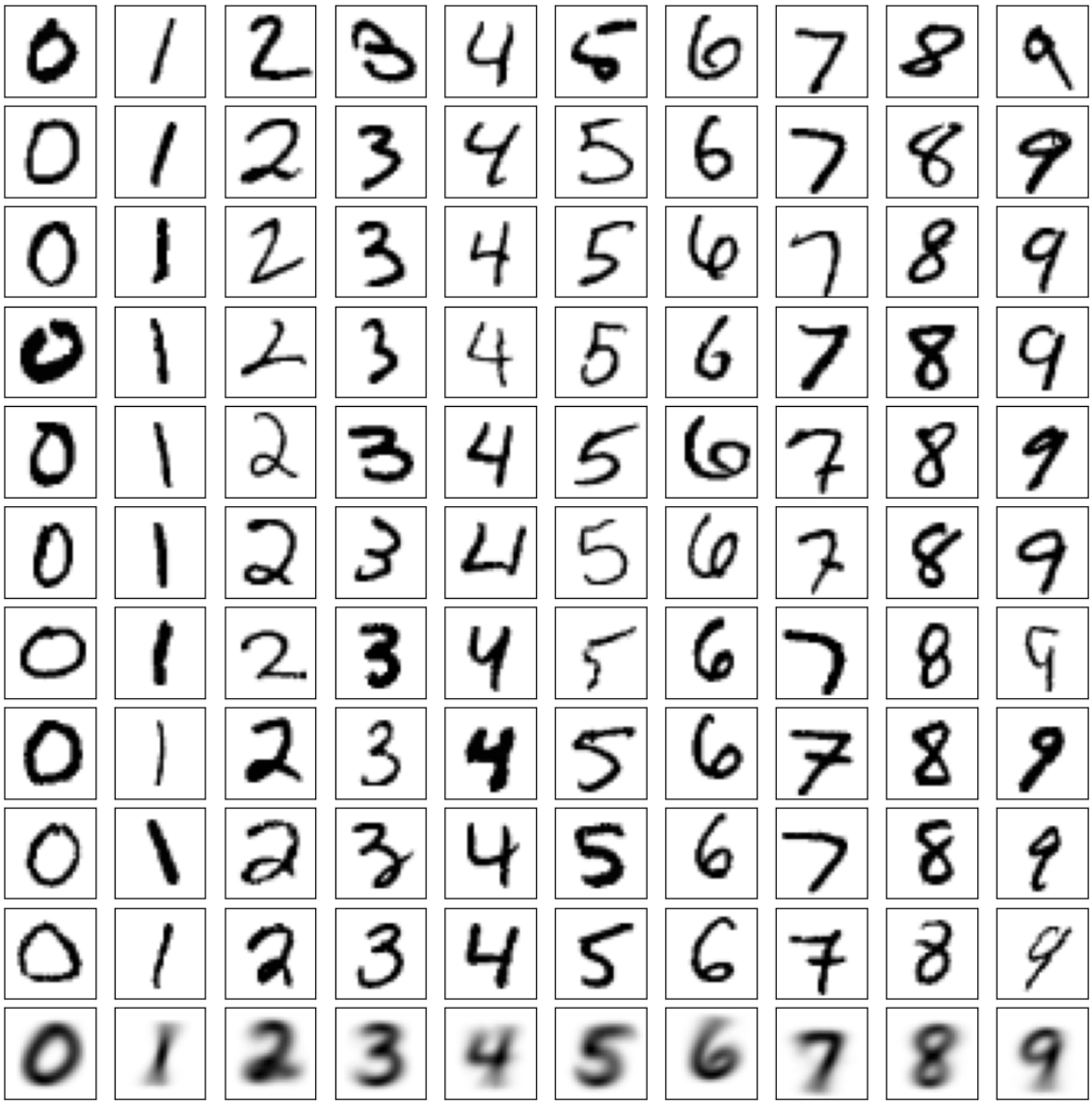


Data of handwritten letters with gray scale

Program: mnist.py

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt

N = 10000
with open('test-images.bin', 'rb') as f1:
    X = np.fromfile(f1, 'uint8', -1)[16:]
X = X.reshape((N, 28, 28))
with open('test-labels.bin', 'rb') as f2:
    Y = np.fromfile(f2, 'uint8', -1)[8:]
D = {y: [] for y in set(Y)}
for x, y in zip(X, Y):
    D[y].append(x)
print([len(D[y]) for y in sorted(D)])

fig, ax = plt.subplots(11, 10, figsize=(10, 10))
plt.subplots_adjust(wspace=0.1, hspace=0.1,
                    left=0.01, right=0.99, bottom=0.01, top=0.99)
for y in D:
    for k in range(10):
        A = 255 - D[y][k]
        ax[k][y].imshow(A, 'gray')
        ax[k][y].tick_params(labelbottom=False, labelleft=False,
                             color='white')
    A = 255 - sum([x.astype('float') for x in D[y]]) / len(D[y])
    ax[10][y].imshow(A, 'gray')
    ax[10][y].tick_params(labelbottom=False, labelleft=False,
                          color='white')
```

```
[980, 1135, 1032, 1010, 982, 892, 958, 1028, 974, 1009]
```



In [ ]: