略解と解説

第1章

発展課題

六芒星の描画。

```
img = Image.new("L", (256, 256), 255)
draw = ImageDraw.Draw(img)
cx = 128
cy = 128
r = 96
draw.ellipse((cx - r, cy - r, cx + r, cy + r))
N = 3
s = 2 * pi / N
for i in range(N):
    s1 = s*i - 0.5 * pi
    s2 = s1 + s * k
    x1 = r * cos(s1) + cx
    y1 = r * sin(s1) + cy
    x2 = r * cos(s2) + cx
    y2 = r * sin(s2) + cy
    draw.line((x1, y1, x2, y2))
    s1 = s*(i+0.5) - 0.5 * pi
    s2 = s1 + s * k
    x1 = r * cos(s1) + cx
    y1 = r * sin(s1) + cy
    x2 = r * cos(s2) + cx
    y2 = r * sin(s2) + cy
    draw.line((x1, y1, x2, y2))
```

第2章

課題2

```
def plot(draw, s):
    hs = s // 2
    red = (255, 0, 0)
    green = (0, 255, 0)
    blue = (0, 0, 255)
    for x in range(s):
        for y in range(s):
            z = complex(x - hs + 0.5, -y + hs + 0.5) / s * 4
            z = newton(z)
            # ここを埋めよ
            if z.real > 0.0:
              c = red
```

```
            else:
              if z.imag > 0.0:
                c = green
              else:
                c = blue
            draw.rectangle([x, y, x + 1, y + 1], fill=c)
```

発展課題

```
def newton(x):
    for _ in range(10):
        x = x - (x**4 - 1) / (4 * x**3)
    return x

def plot(draw, s):
    hs = s // 2
    red = (255, 0, 0)
    green = (0, 255, 0)
    blue = (0, 0, 255)
    purple = (255, 0 ,255)
    for x in range(s):
        for y in range(s):
            z = complex(x - hs + 0.5, -y + hs + 0.5) / s * 4
            z = newton(z)
            # ここを埋めよ
            if z.real + z.imag > 0.0:
              if z.real - z.imag > 0.0:
                c = red
              else:
                c = purple
            else:
              if z.real - z.imag > 0.0:
                c = green
              else:
                c = blue
            draw.rectangle([x, y, x + 1, y + 1], fill=c)
```

第3章

課題2

```
def collatz(i):
    print(i)
    while (i!=1):
        if (i%2==0):
            i = i // 2
        else:
            i = i * 3 + 1
        print(i)
```

```
```

発展課題

```
def collatz_graph(i, edges):
    while (i!=1 and i!=3):
        j = i
        if (i%2==0):
            i = i // 2
        else:
            i = i * 3 + 3
        edges.add((j, i))
```

第7章

課題1-1

```
def kaidan(n):
    # 終端条件
    if n==1:
        return 1
    if n==2:
        return 2
    # 再帰部分
    return kaidan(n-1)+kaidan(n-2)
```

課題2-1

```
def solve(x, y, step, maze):
    if maze[x][y] == '*':
        return
    if isinstance(maze[x][y], int):
        return
    maze[x][y] = step
    solve(x+1, y, step+1, maze) # 右を探索
    # 残りを埋めよ
    solve(x-1, y, step+1, maze)
    solve(x, y+1, step+1, maze)
    solve(x, y-1, step+1, maze)
```

第9章

そのまま実行すると、以下のように画像のところどころに黒い点が現れる「黒飛び」が発生する。

近似画像の「黒飛び」が発生するのは、画像のピクセルの値が0から255の範囲から超えてしまうためだ。関数svdにおいて、近似値を構成しているところで、以下のように最小値と最大値を指定してやると「黒飛び」がなくなる。

```
    b = np.asarray(ur * sr * vr)
    b = np.clip(b, 0,255)        # 最小値と最大値を指定
```

講義では「なぜ黒飛びが発生するか」を考察させると良い。

第10章

課題2-2

電卓に乗除算を追加する。

```
def calc(code):
    data = code.split()
    stack = []
    for x in data:
        print(stack, x, end=" => ")
        if x == '+':
            b = stack.pop()
            a = stack.pop()
            stack.append(a+b)
        elif x == '-':
            b = stack.pop()
```

```
            a = stack.pop()
            stack.append(a-b)
        # 以下を追加
        elif x == '*':
            b = stack.pop()
            a = stack.pop()
            stack.append(a*b)
        elif x == '/':
            b = stack.pop()
            a = stack.pop()
            stack.append(a//b)
        else:
            stack.append(int(x))
        print(stack)
    print(stack.pop())
```

発展課題

例えば「1 + 2 * 3 - 4」の例であれば、まず以下を実行する。

```
dis.dis("a + b * c - d")
```

すると、以下のような結果が得られる。

```
  1           0 LOAD_NAME                0 (a)
              2 LOAD_NAME                1 (b)
              4 LOAD_NAME                2 (c)
              6 BINARY_MULTIPLY
              8 BINARY_ADD
             10 LOAD_NAME                3 (d)
             12 BINARY_SUBTRACT
             14 RETURN_VALUE
```

これを見ながら、逆ポーランド記法を組み立てる。`LOAD_NAME`はそのまま値を、`BINARY_MULTPLY`は乗算記号`*`をと書いていくと`1 2 3 * + 4 -`という文字列を作ることができる。「電卓」に入力してみると

```
calc("1 2 3 * + 4 -")
```

```
[] 1 => [1]
[1] 2 => [1, 2]
[1, 2] 3 => [1, 2, 3]
[1, 2, 3] * => [1, 6]
[1, 6] + => [7]
[7] 4 => [7, 4]
```

```
[7, 4] - => [3]
3
```

と、正しい答え3が得られる。「(1 + 2 * 3) / 4」の場合も同様。

第12章

課題1

最初に選んだ箱が正解だった場合は、残りの箱からランダムに選ぶので、

```
        second_choice = choice(rest_boxes) # ここを埋めよ(1)
```

また、最初に選んだ箱が正解ではなかった場合は、次に選ぶ箱は必ず正解になるので

```
        second_choice = answer # ここを埋めよ(2)
```

課題2

左から右に通行可能な確率は、pが0.5より小さいとほとんどゼロ、0.5より大きいとほぼ1という、ステップ関数のような関数になる。このように、パラメタを変えた時、あるところで系の状態が劇的に変化する現象を相転移と呼ぶ。パーコレーションは相転移を起こす最も簡単なモデルの一つである。

発展課題

```
def gacha(n):
    cd = 0
    posters = []
    while len(set(posters)) < n:
        posters.append(random.randint(1, n))
        cd += 1
    return cd
```

第13章

課題2

```
@jit
def laplacian(m, n, s):
    ts = 0.0
    ts += s[m+1][n]
    ts += s[m-1][n]
    ts += s[m][n+1]
    ts += s[m][n-1]
```

```python
        # ここを埋めよ
        ts -= 4*s[m][n]
        return ts
```