Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

A Lab Report on

“Computer Graphics”

[Code No: COMP342]

LAB- IV

Submitted by:

Aakash Thakur

CS III-I

Roll-No: 20

Submitted to:

Mr. Dhiraj Shrestha

Department of Computer Science and Engineering

Submission Date:

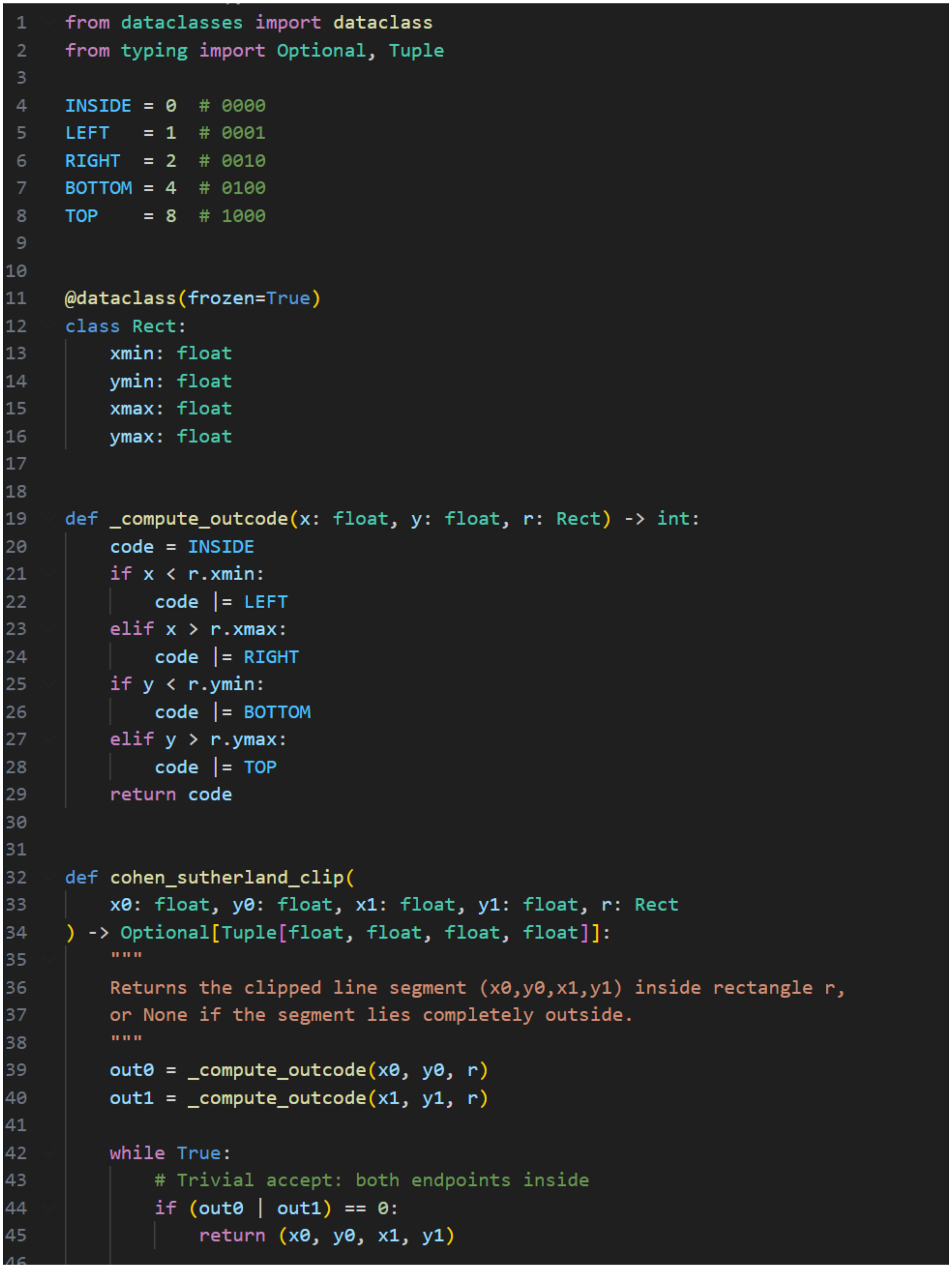
2026/1/13

Implement Cohen Sutherland Line Clipping algorithm

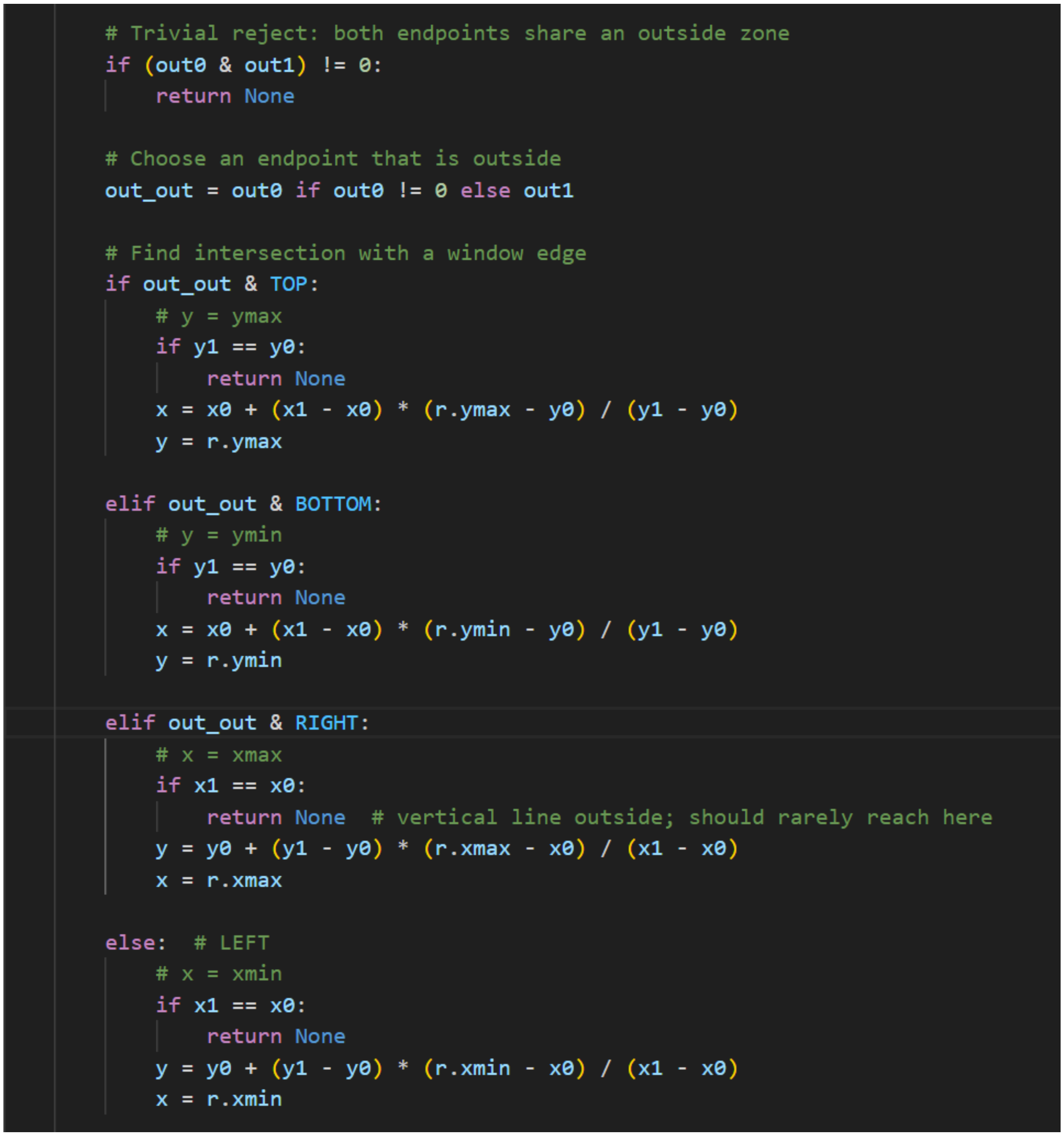
Algorithm:

1. Assign a region code for each endpoints.
2. If both endpoints have a region code 0000 -> trivially accept these line.
3. Else perform the logical AND operation for both region codes.
    a. If the result is NOT 0000 -> trivially reject the line.
    b. Else (result = 0000, need clipping)
        i. Choose an endpoint of the line that is outside the window.
        ii. Find the intersection point at the window boundary (based on region code).
        iii. Replace endpoint with the intersection point and update the region code.
        iv. Repeat step 2 until we find a clipped line either trivially accepted or trivially rejected
4. Repeat step 1 for other lines.

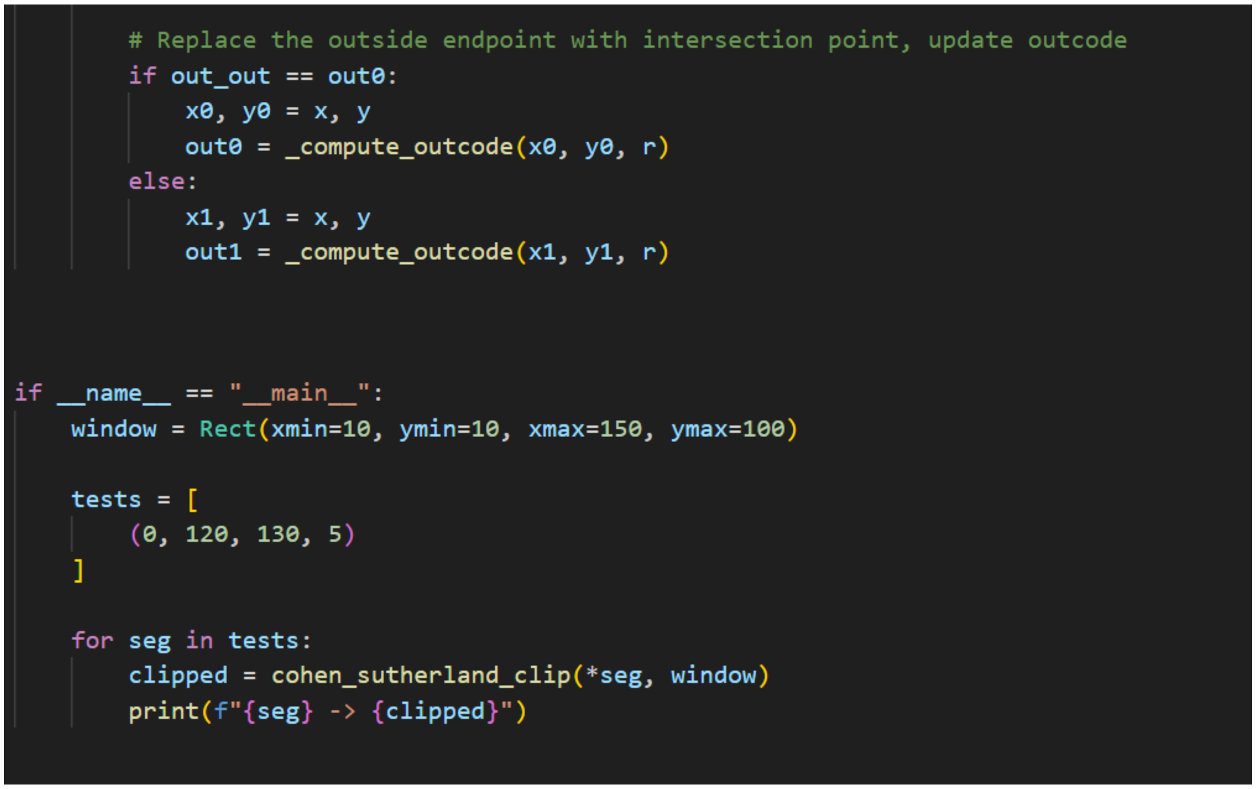
Source Code:



```python
from dataclasses import dataclass
from typing import Optional, Tuple

INSIDE = 0  # 0000
LEFT   = 1  # 0001
RIGHT  = 2  # 0010
BOTTOM = 4  # 0100
TOP    = 8  # 1000


@dataclass(frozen=True)
class Rect:
    xmin: float
    ymin: float
    xmax: float
    ymax: float


def _compute_outcode(x: float, y: float, r: Rect) -> int:
    code = INSIDE
    if x < r.xmin:
        code |= LEFT
    elif x > r.xmax:
        code |= RIGHT
    if y < r.ymin:
        code |= BOTTOM
    elif y > r.ymax:
        code |= TOP
    return code


def cohen_sutherland_clip(
    x0: float, y0: float, x1: float, y1: float, r: Rect
) -> Optional[Tuple[float, float, float, float]]:
    """
    Returns the clipped line segment (x0,y0,x1,y1) inside rectangle r,
    or None if the segment lies completely outside.
    """
    out0 = _compute_outcode(x0, y0, r)
    out1 = _compute_outcode(x1, y1, r)

    while True:
        # Trivial accept: both endpoints inside
        if (out0 | out1) == 0:
            return (x0, y0, x1, y1)
```

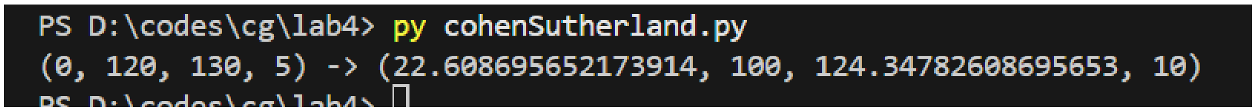


```python
    # Trivial reject: both endpoints share an outside zone
    if (out0 & out1) != 0:
        return None

    # Choose an endpoint that is outside
    out_out = out0 if out0 != 0 else out1

    # Find intersection with a window edge
    if out_out & TOP:
        # y = ymax
        if y1 == y0:
            return None
        x = x0 + (x1 - x0) * (r.ymax - y0) / (y1 - y0)
        y = r.ymax

    elif out_out & BOTTOM:
        # y = ymin
        if y1 == y0:
            return None
        x = x0 + (x1 - x0) * (r.ymin - y0) / (y1 - y0)
        y = r.ymin

    elif out_out & RIGHT:
        # x = xmax
        if x1 == x0:
            return None  # vertical line outside; should rarely reach here
        y = y0 + (y1 - y0) * (r.xmax - x0) / (x1 - x0)
        x = r.xmax

    else:  # LEFT
        # x = xmin
        if x1 == x0:
            return None
        y = y0 + (y1 - y0) * (r.xmin - x0) / (x1 - x0)
        x = r.xmin
```



```
            # Replace the outside endpoint with intersection point, update outcode
            if out_out == out0:
                x0, y0 = x, y
                out0 = _compute_outcode(x0, y0, r)
            else:
                x1, y1 = x, y
                out1 = _compute_outcode(x1, y1, r)


if __name__ == "__main__":
    window = Rect(xmin=10, ymin=10, xmax=150, ymax=100)

    tests = [
        (0, 120, 130, 5)
    ]

    for seg in tests:
        clipped = cohen_sutherland_clip(*seg, window)
        print(f"{seg} -> {clipped}")
```

Output:



```
PS D:\codes\cg\lab4> py cohenSutherland.py
(0, 120, 130, 5) -> (22.608695652173914, 100, 124.34782608695653, 10)
```

Implement Liang Barsky Line Clipping algorithm

Algorithm:

- $x_{min}, y_{min}, x_{max}, y_{max}$
  and a line segment from $P_0(x_0, y_0)$to $P_1(x_1, y_1)$.

1. Compute:
   - $\Delta x = x_1 - x_0$
   - $\Delta y = y_1 - y_0$

2. Represent the line parametrically:
    - $x(t) = x_0 + t\Delta x$
    - $y(t) = y_0 + t\Delta y$
    - where $t \in [0,1]$
3. Initialize the entering and leaving parameters:
    - $t_{enter} = 0$
    - $t_{leave} = 1$
4. For each window boundary, form $(p, q)$and update $[t_{enter}, t_{leave}]$:
    - Left boundary $x \geq x_{min}$: $p = -\Delta x,\ q = x_0 - x_{min}$
    - Right boundary $x \leq x_{max}$: $p = \Delta x,\ q = x_{max} - x_0$
    - Bottom boundary $y \geq y_{min}$: $p = -\Delta y,\ q = y_0 - y_{min}$
    - Top boundary $y \leq y_{max}$: $p = \Delta y,\ q = y_{max} - y_0$

For each boundary:

a. If $p = 0$:

- If $q < 0$→ line is parallel and outside → **trivially reject**
- Else → line is parallel and inside for this boundary → **continue**

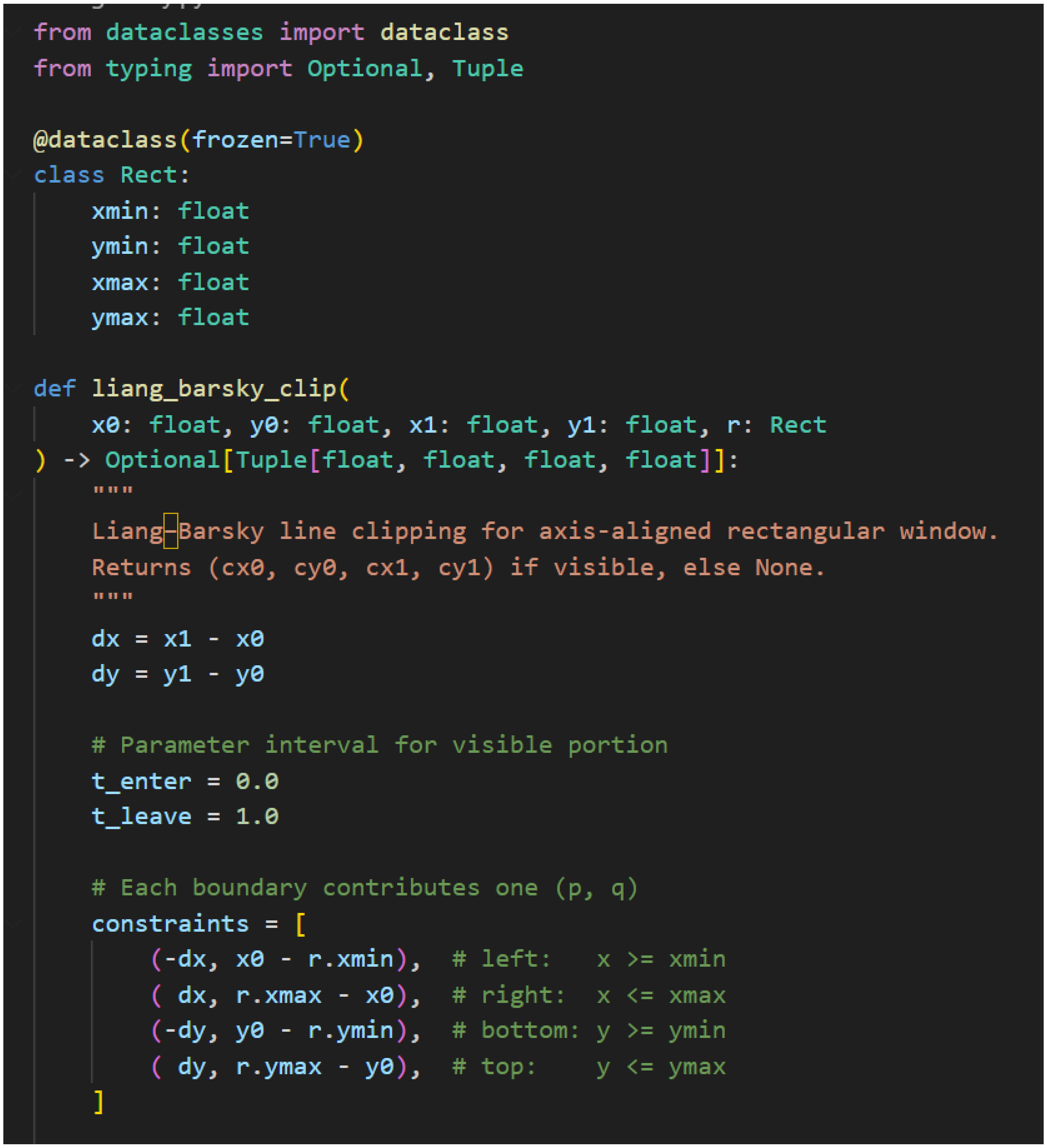
b. Else compute $r = q/p$:

- If $p < 0$(potentially entering):
    - $t_{enter} = \max(t_{enter}, r)$
- If $p > 0$(potentially leaving):
    - $t_{leave} = \min(t_{leave}, r)$

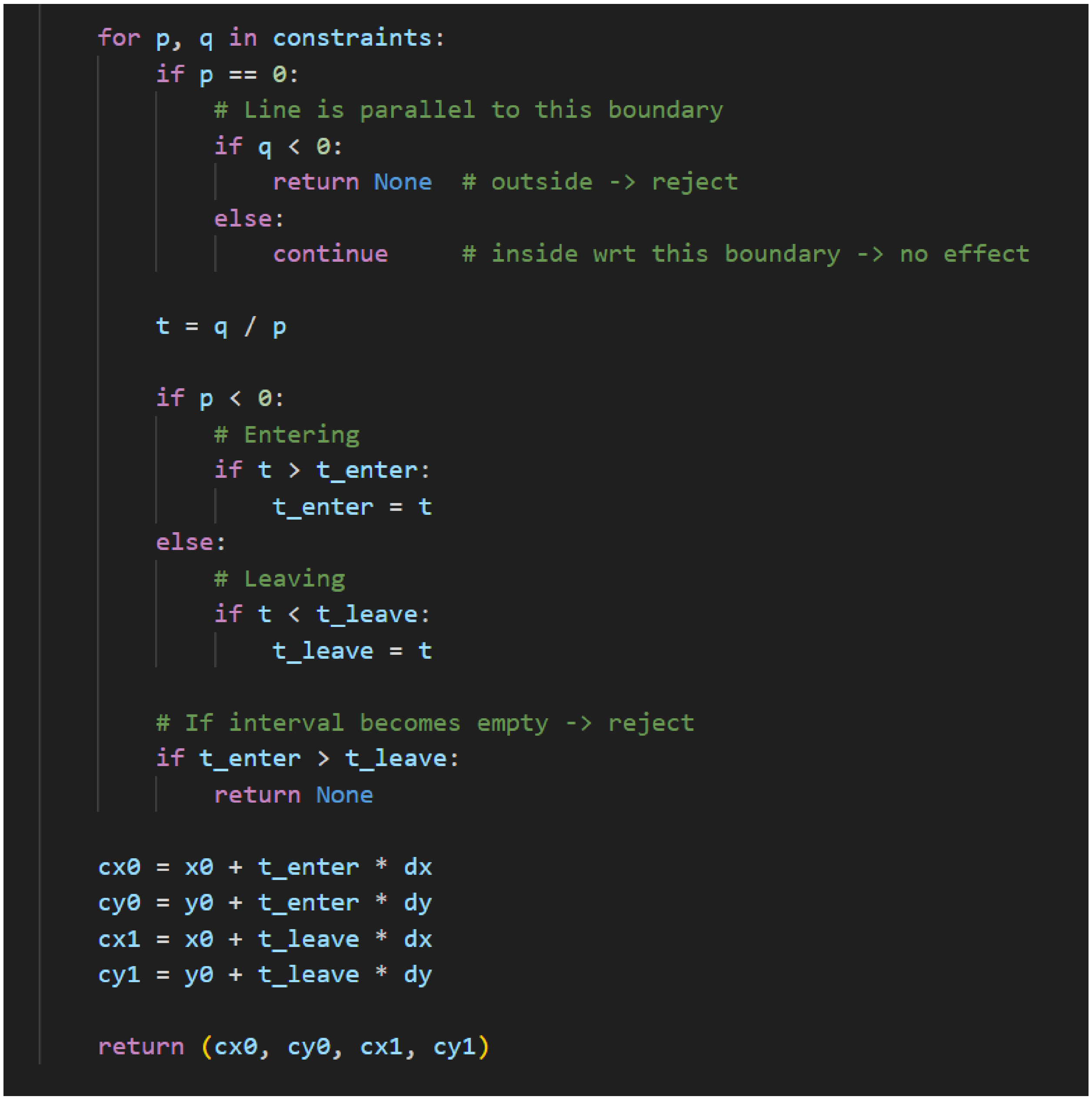
c. If at any point $t_{enter} > t_{leave}$→ **reject**

5. If not rejected, compute clipped endpoints:
    - $P_0' = (x(t_{enter}), y(t_{enter}))$
    - $P_1' = (x(t_{leave}), y(t_{leave}))$
6. Output the clipped line segment.

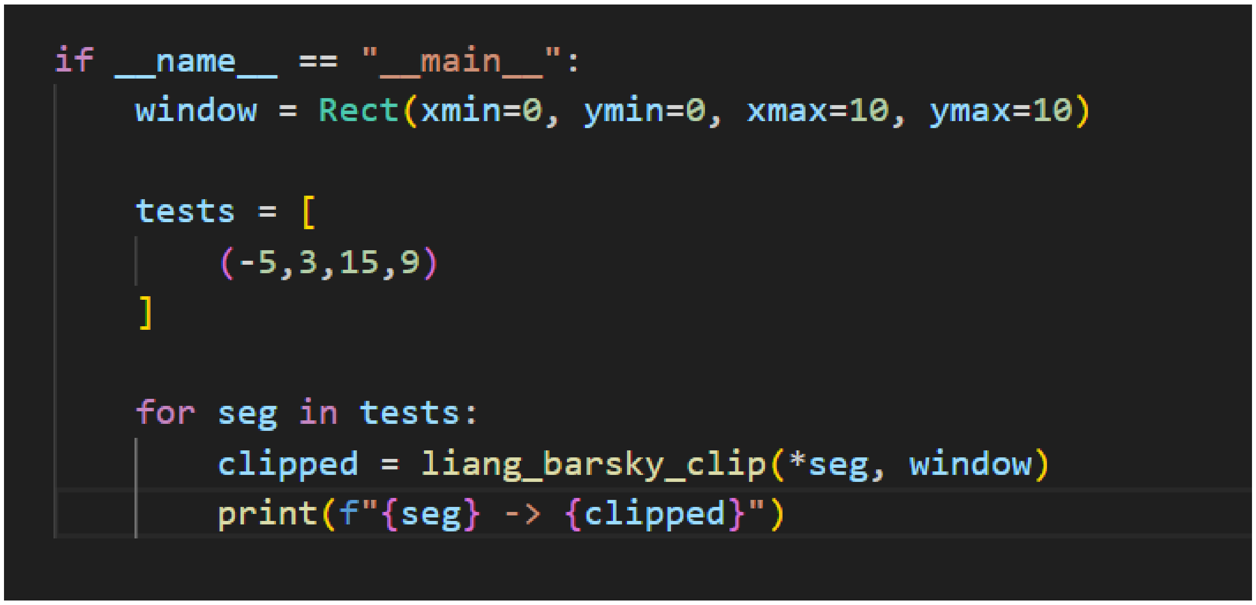
Source Code:



```python
from dataclasses import dataclass
from typing import Optional, Tuple

@dataclass(frozen=True)
class Rect:
    xmin: float
    ymin: float
    xmax: float
    ymax: float

def liang_barsky_clip(
    x0: float, y0: float, x1: float, y1: float, r: Rect
) -> Optional[Tuple[float, float, float, float]]:
    """
    Liang-Barsky line clipping for axis-aligned rectangular window.
    Returns (cx0, cy0, cx1, cy1) if visible, else None.
    """
    dx = x1 - x0
    dy = y1 - y0

    # Parameter interval for visible portion
    t_enter = 0.0
    t_leave = 1.0

    # Each boundary contributes one (p, q)
    constraints = [
        (-dx, x0 - r.xmin),  # left:   x >= xmin
        ( dx, r.xmax - x0),  # right:  x <= xmax
        (-dy, y0 - r.ymin),  # bottom: y >= ymin
        ( dy, r.ymax - y0),  # top:    y <= ymax
    ]
```

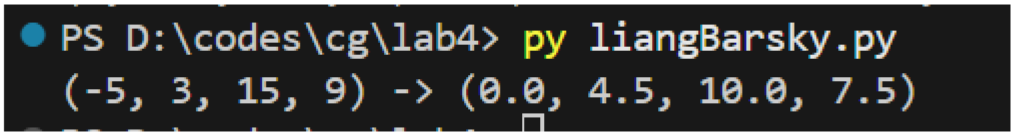


```python
    for p, q in constraints:
        if p == 0:
            # Line is parallel to this boundary
            if q < 0:
                return None  # outside -> reject
            else:
                continue     # inside wrt this boundary -> no effect

        t = q / p

        if p < 0:
            # Entering
            if t > t_enter:
                t_enter = t
        else:
            # Leaving
            if t < t_leave:
                t_leave = t

        # If interval becomes empty -> reject
        if t_enter > t_leave:
            return None

    cx0 = x0 + t_enter * dx
    cy0 = y0 + t_enter * dy
    cx1 = x0 + t_leave * dx
    cy1 = y0 + t_leave * dy

    return (cx0, cy0, cx1, cy1)
```



```python
if __name__ == "__main__":
    window = Rect(xmin=0, ymin=0, xmax=10, ymax=10)

    tests = [
        (-5,3,15,9)
    ]

    for seg in tests:
        clipped = liang_barsky_clip(*seg, window)
        print(f"{seg} -> {clipped}")
```

Output:



```
PS D:\codes\cg\lab4> py liangBarsky.py
(-5, 3, 15, 9) -> (0.0, 4.5, 10.0, 7.5)
```

Implement Sutherland Hodgemann polygon clipping algorithm.

Algorithm:

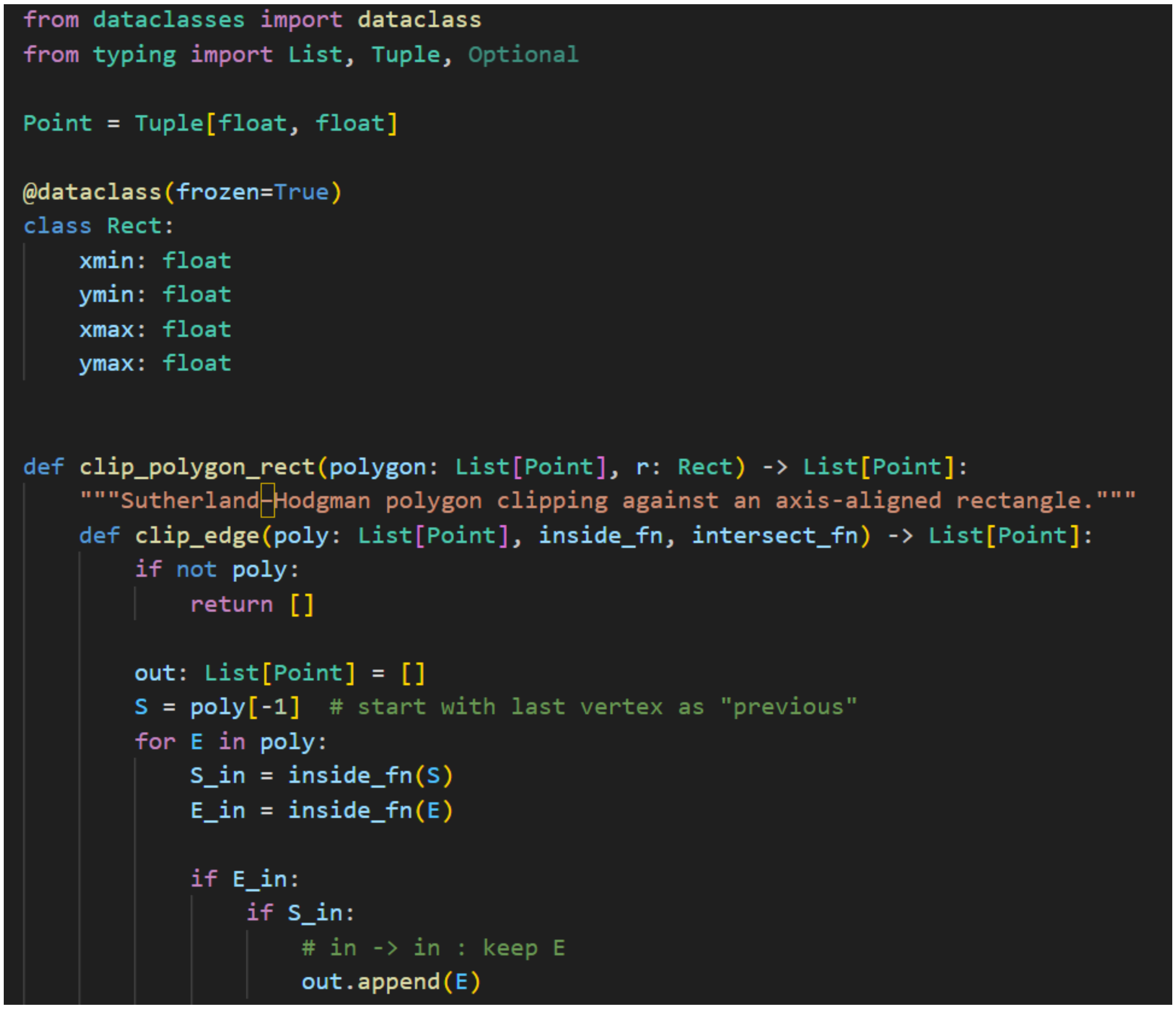
For each edge of the clipping window (one edge at a time):

1. Take the current polygon's vertex list as the **input list**.
2. Start with an empty **output list**.
3. For each polygon edge formed by consecutive vertices **S → E** (Start → End):
   - If **E is inside** the clip edge:
     - If **S is inside**: output **E**

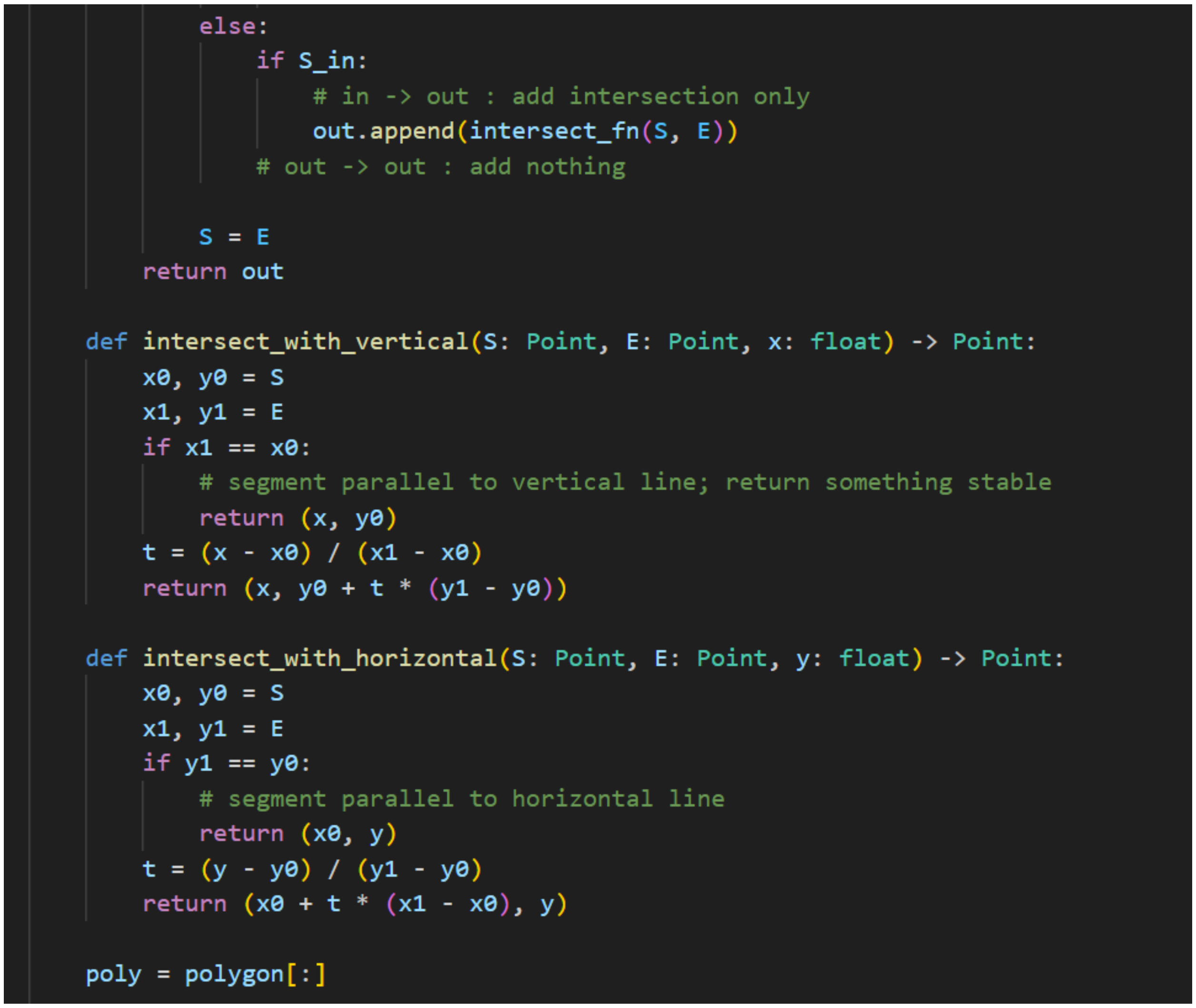
        - Else (**S outside**): output **intersection(S,E)** then **E**
    - Else (**E outside**):
        - If **S is inside**: output **intersection(S,E)**
        - Else (**S outside**): output nothing

4. After processing all S→E pairs, the output list becomes the polygon for the next clip edge.
5. After all clip edges are processed, the remaining polygon is the clipped result (may be empty).

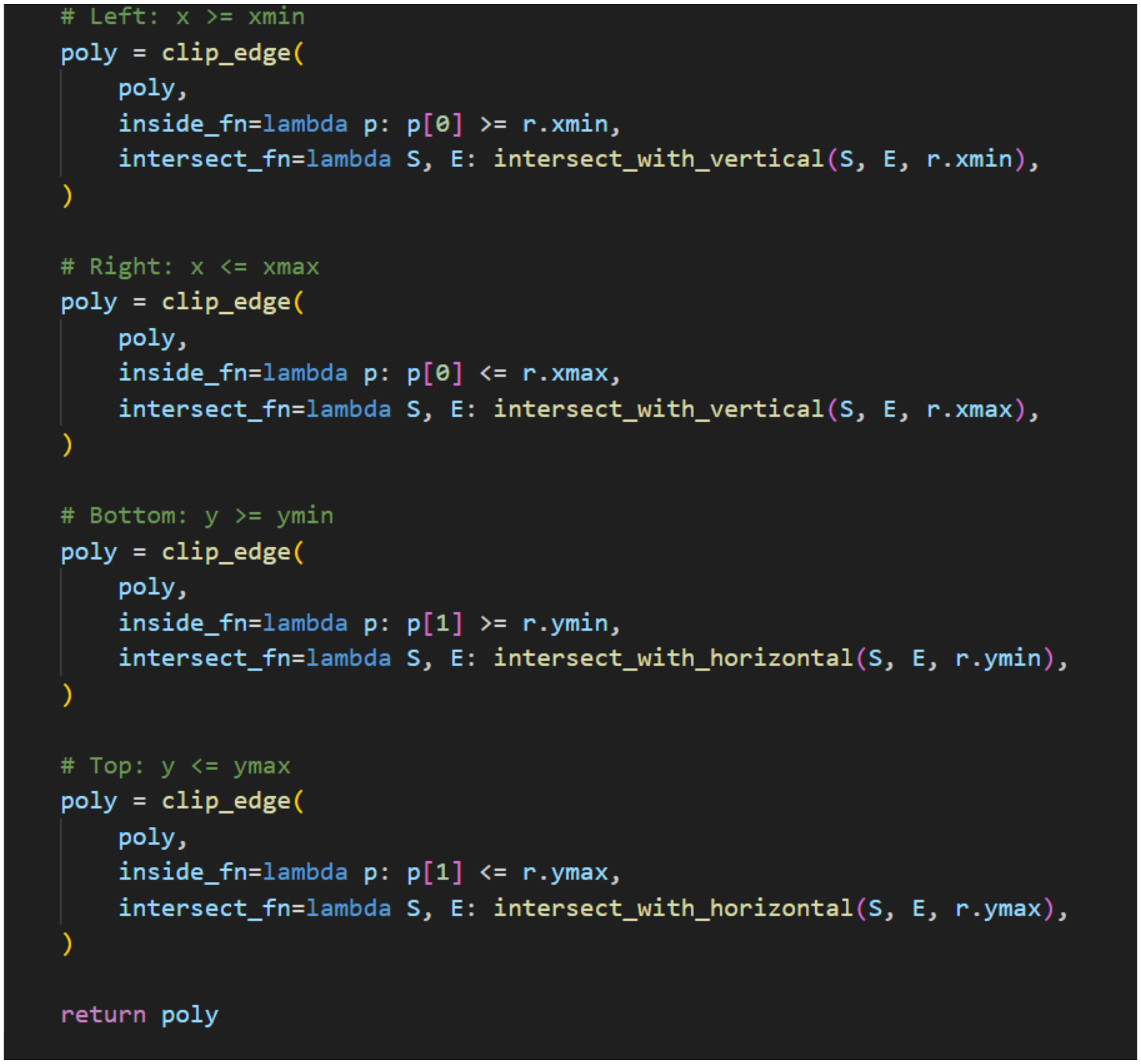
Source Code:



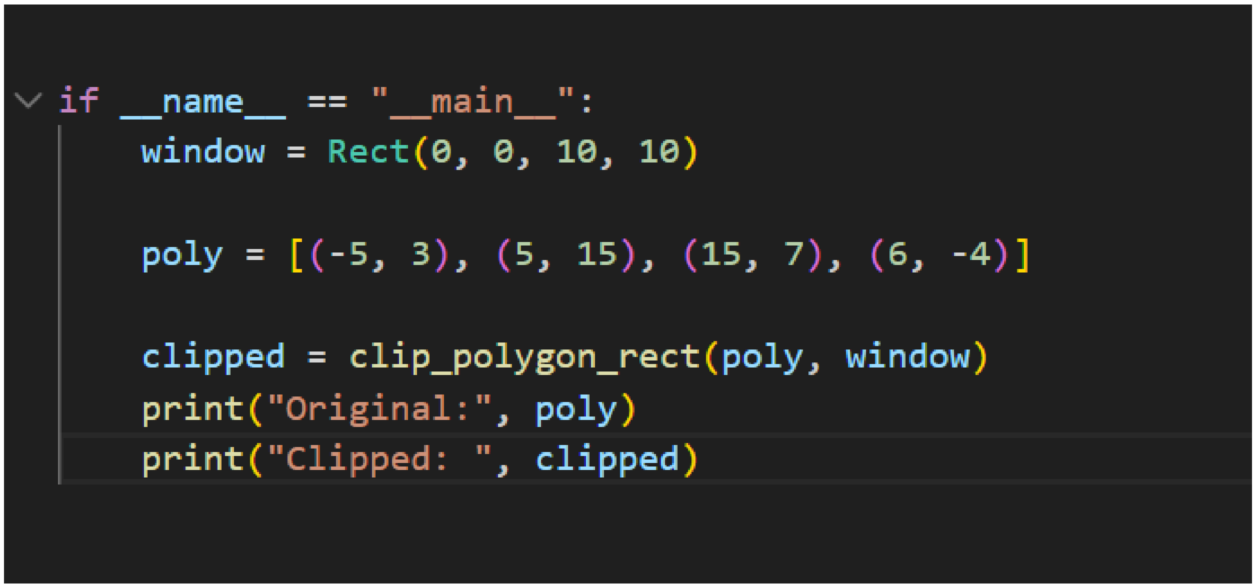
```python
from dataclasses import dataclass
from typing import List, Tuple, Optional

Point = Tuple[float, float]

@dataclass(frozen=True)
class Rect:
    xmin: float
    ymin: float
    xmax: float
    ymax: float


def clip_polygon_rect(polygon: List[Point], r: Rect) -> List[Point]:
    """Sutherland–Hodgman polygon clipping against an axis-aligned rectangle."""
    def clip_edge(poly: List[Point], inside_fn, intersect_fn) -> List[Point]:
        if not poly:
            return []

        out: List[Point] = []
        S = poly[-1]  # start with last vertex as "previous"
        for E in poly:
            S_in = inside_fn(S)
            E_in = inside_fn(E)

            if E_in:
                if S_in:
                    # in -> in : keep E
                    out.append(E)
```



```python
        else:
            if S_in:
                # in -> out : add intersection only
                out.append(intersect_fn(S, E))
            # out -> out : add nothing

        S = E
    return out

def intersect_with_vertical(S: Point, E: Point, x: float) -> Point:
    x0, y0 = S
    x1, y1 = E
    if x1 == x0:
        # segment parallel to vertical line; return something stable
        return (x, y0)
    t = (x - x0) / (x1 - x0)
    return (x, y0 + t * (y1 - y0))

def intersect_with_horizontal(S: Point, E: Point, y: float) -> Point:
    x0, y0 = S
    x1, y1 = E
    if y1 == y0:
        # segment parallel to horizontal line
        return (x0, y)
    t = (y - y0) / (y1 - y0)
    return (x0 + t * (x1 - x0), y)

poly = polygon[:]
```

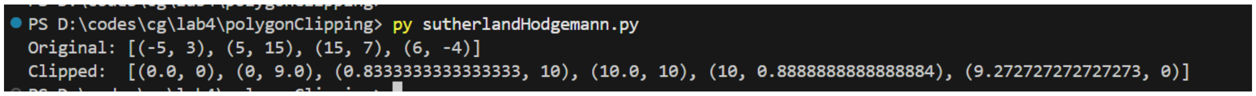


```
    # Left: x >= xmin
    poly = clip_edge(
        poly,
        inside_fn=lambda p: p[0] >= r.xmin,
        intersect_fn=lambda S, E: intersect_with_vertical(S, E, r.xmin),
    )

    # Right: x <= xmax
    poly = clip_edge(
        poly,
        inside_fn=lambda p: p[0] <= r.xmax,
        intersect_fn=lambda S, E: intersect_with_vertical(S, E, r.xmax),
    )

    # Bottom: y >= ymin
    poly = clip_edge(
        poly,
        inside_fn=lambda p: p[1] >= r.ymin,
        intersect_fn=lambda S, E: intersect_with_horizontal(S, E, r.ymin),
    )

    # Top: y <= ymax
    poly = clip_edge(
        poly,
        inside_fn=lambda p: p[1] <= r.ymax,
        intersect_fn=lambda S, E: intersect_with_horizontal(S, E, r.ymax),
    )

    return poly
```



```python
if __name__ == "__main__":
    window = Rect(0, 0, 10, 10)

    poly = [(-5, 3), (5, 15), (15, 7), (6, -4)]

    clipped = clip_polygon_rect(poly, window)
    print("Original:", poly)
    print("Clipped: ", clipped)
```

Output:



```
PS D:\codes\cg\lab4\polygonClipping> py sutherlandHodgemann.py
Original: [(-5, 3), (5, 15), (15, 7), (6, -4)]
Clipped:  [(0.0, 0), (0, 9.0), (0.8333333333333333, 10), (10.0, 10), (10, 0.8888888888888884), (9.272727272727273, 0)]
```