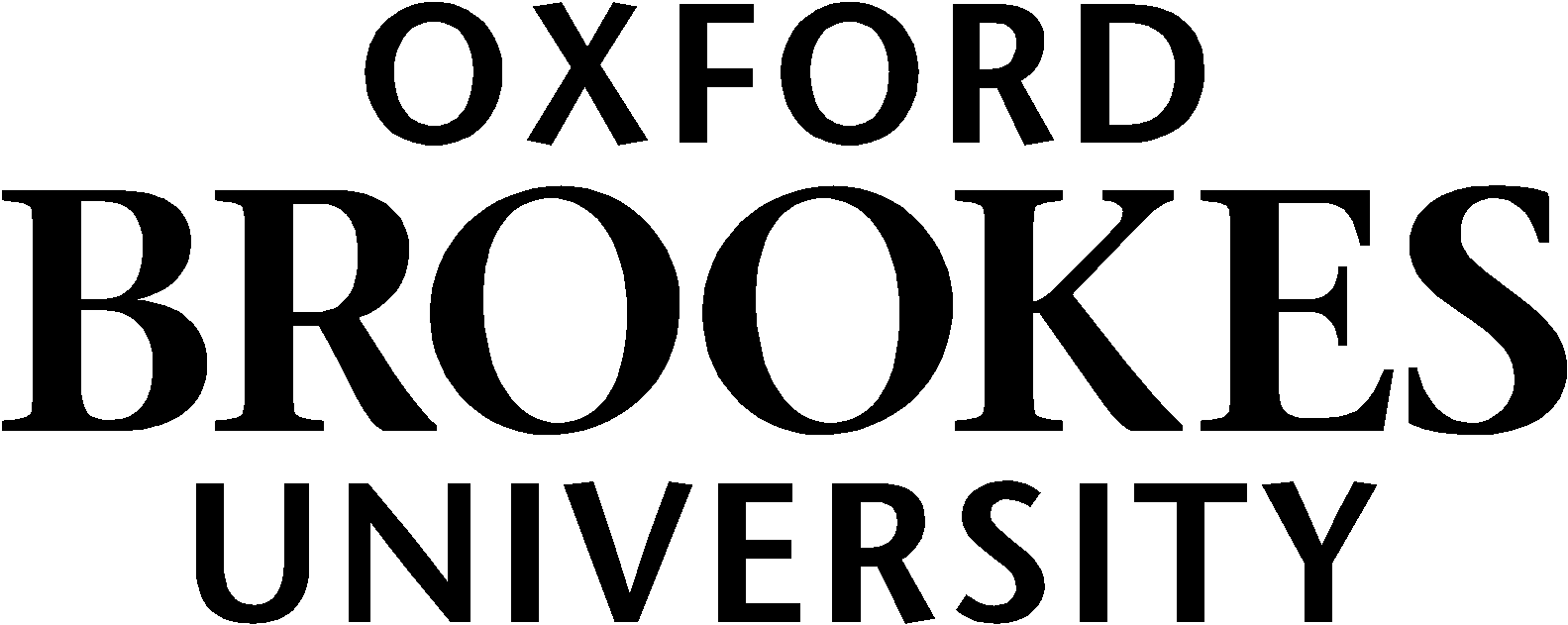
****

**Assessment cover**

|  |  |  |  |
| --- | --- | --- | --- |
| Module No: | COMP4004 | Module title: | **Problem Solving and Programming** |

|  |  |
| --- | --- |
| Assessment title : | **End of Semester Assessment** |

|  |  |
| --- | --- |
| Due date and time**:** | **13th December 2024** |

|  |  |
| --- | --- |
| Estimated total time to be spent on assignment: | 40 hours per student |

**LEARNING OUTCOMES**

|  |
| --- |
| **On successful completion of this assignment, students will be able to achieve the module following learning outcomes (LOs):** *LO numbers and text copied and pasted from the module descriptor* |
| *LO 1: Design, implement and test computer programs, derived from application based case studies, implemented using an appropriate computer programming language* |
| *LO 2: Demonstrate a thorough understanding of the fundamental concepts of high level programming languages, including the syntax and semantics of constructs for input/output, control ﬂow and elementary data structuring.* |
|  |

|  |  |  |
| --- | --- | --- |
| **Engineering Council AHEP4 LOs assessed (from S2 2022-23)**  *LOs copied and pasted from the AHEP4 matrix (add rows as required)* | |  |
| **LO number** | **LO text** | **Met? (Y/N)** |
| **C3** | Select and apply appropriate computational and analytical techniques to model complex problems, recognising the limitations of the techniques employed |  |
| **C4** | Select and evaluate technical literature and other sources of information to address complex problems |  |
|  |  |  |

The use of AI tools is not allowed

**Statement of Compliance *(please tick to sign)***

I declare that the work submitted is my own and that the work I submit is fully in accordance with the University regulations regarding assessments *(*[*www.brookes.ac.uk/uniregulations/current*](http://www.brookes.ac.uk/uniregulations/current)*)*

from tkinter import \*

root = Tk()

canvas = Canvas(root,width = 1000, height = 700, bg = 'white')

def read\_shapes(filename):

    """

    Reads file and returns a corresponding list of shapes

    One line for each shape

    Args:

        filename (str): File with shape values

    Returns:

        shapes(list): List of shape dictionaries

    """

    shapes = []

    with open(filename, 'r', encoding="utf-8") as file:

        for line in file:

            # Splits the every line in file

            # And assigns a variable to each value

            a = line.split()

            kindval = a[0]

            x0, y0, x1, y1 = map(int, a[1:5])

            fillval = a[5] == 'True'

            # Adds said values in a dictionary

            # And appends dictionaries in list 'shapes'

            shape = {'kind': kindval, 'bounds': [x0, y0, x1, y1], 'fill': fillval}

            shapes.append(shape)

        return shapes

def draw\_on\_canvas(can, shape):

    """

    Assigns values to each variable from

    List or list of dictionaries

    Args:

        can (tkinter.Canvas): White canvas

        shape (dict): Shapes to be printed on the canvas

    """

    if isinstance(shape, dict):

        shape = [shape]

    for s in shape:

        kind = s['kind']

        bounds = s['bounds']

        fill = s['fill']

        # Rectangle creation (colour, vertex, summits)

        if kind == 'rect':

            x0, y0, x1, y1 = bounds

            fill\_colour = 'black' if fill else 'white'

            outline\_colour = 'black' if fill else 'white'

            can.create\_rectangle(x0, y0, x1, y1, fill=fill\_colour, outline = outline\_colour)

        # Triangle creation (colour, vertex, summits)

        elif kind == 'tri':

            x0, y0, x1, y1 = bounds

            fill\_colour = 'black' if fill else 'white'

            outline\_colour = 'black' if fill else 'white'

            can.create\_polygon(x0, y0, x1, y0, x0, y1, fill=fill\_colour, outline = outline\_colour)

def translate(shape, xtrans):

    """

    Translates a shape horizontally by a set amount

    Args:

        shape (dict): Shape to be translated

        xtrans (int): The amount to translate

    Returns:

        tshape (dict): Returns the translated shape

    """

    kind = shape['kind']

    bounds = shape['bounds']

    fill = shape['fill']

    x0, y0, x1, y1 = bounds

    tx0 = x0+xtrans

    tx1 = x1+xtrans

    tshape = {'bounds': [tx0, y0, tx1, y1], 'kind': kind, 'fill': fill}

    return tshape

def reflect\_v(shape, xval):

    """

    Reflects a shape over a defined y-axis

    Args:

        shape (dict): Shape to be reflected

        xval (int): x value of the axis of symmetry

    Returns:

        vshape (dict): Returns the reflected shape

    """

    kind = shape['kind']

    bounds = shape['bounds']

    fill = shape['fill']

    x0, y0, x1, y1 = bounds

    vx0 = (2\*xval) - x0

    vx1 = (2\*xval) - x1

    vshape = {'bounds': [vx0, y0, vx1, y1], 'kind': kind, 'fill': fill}

    return vshape

def reflect\_h(shape, yval):

    """

    Reflects a shape over a defined x-axis

    Args:

        shape (dict): Shape to be reflected

        yval (int): y value of the axis of symmetry

    Returns:

        hshape(dict): Returns the reflected shape

    """

    kind = shape['kind']

    bounds = shape['bounds']

    fill = shape['fill']

    x0, y0, x1, y1 = bounds

    hy0 = (2\*yval) - y0

    hy1 = (2\*yval) - y1

    hshape = {'bounds': [x0, hy0, x1, hy1], 'kind': kind, 'fill': fill}

    return hshape

def rotate(shape, xcentre, ycentre):

    """

    Rotates a shape over a defined center of symmetry

    Args:

        shape (dict): Shape to be reflected

        xcentre (int): x value of the center of symmetry

        ycentre (int): y value of the center of symmetry

    Returns:

        rshape(dict): Returns the rotated shape

    """

    kind = shape['kind']

    bounds = shape['bounds']

    fill = shape['fill']

    x0, y0, x1, y1 = bounds

    rx0 = (2\*xcentre) - x0

    rx1 = (2\*xcentre) - x1

    ry0 = (2\*ycentre) - y0

    ry1 = (2\*ycentre) - y1

    rshape = {'bounds': [rx0, ry0, rx1, ry1], 'kind': kind, 'fill': fill}

    return rshape

def glide\_reflect(shape,xtrans,yval):

    """

    Translates a shape horizontally by a set amount

    and reflects it over a set x-axis

    Args:

        shape (dict): Shape to be translated and reflected

        xtrans (int): The amount to translate

        yval (int): y value of the axis of symmetry

    Returns:

        gshape(dict): Returns the translated and reflected shape

    """

    kind = shape['kind']

    bounds = shape['bounds']

    fill = shape['fill']

    x0, y0, x1, y1 = bounds

    gx0 = x0+xtrans

    gx1 = x1+xtrans

    gy0 = (2\*yval) - y0

    gy1 = (2\*yval) - y1

    gshape = {'bounds': [gx0, gy0, gx1, gy1], 'kind': kind, 'fill': fill}

    return gshape

def make\_frieze(filename, frieze\_group, repeat\_length, height, nbr\_repeats):

    """Displays a frieze by taking the shapes in a text file

        applying various operations to it in order to create a pattern,

        and then constructing set number of repeats of that pattern

    Args:

        filename (str): Text file containing the shapes dictionaries in order to make a frieze

        frieze\_group (int): An integer, ranging from 1 to 7,

            that describes the symmetry of a frieze to be created

        repeat\_length (int): The distance between occurrences of a repeating pattern

        height (int): The height of the frieze

        nbr\_repeats (int): The number of times that the pattern should repeat

    """

    foot\_shape = read\_shapes(filename)

    for i in range(nbr\_repeats):

        for shape in foot\_shape:

            t\_shape = translate(shape, i \* repeat\_length)

            v\_shape = reflect\_v(shape, (i+1) \* repeat\_length//2)

            r\_shape = rotate(shape, (i+1)\*repeat\_length//2, height//2)

            if frieze\_group == 1:

                draw\_on\_canvas(canvas, t\_shape)

            elif frieze\_group == 2:

                g\_shape = glide\_reflect(t\_shape, repeat\_length//2, height//2)

                draw\_on\_canvas(canvas, t\_shape)

                draw\_on\_canvas(canvas, g\_shape)

            elif frieze\_group == 3:

                draw\_on\_canvas(canvas, t\_shape)

                draw\_on\_canvas(canvas, v\_shape)

            elif frieze\_group == 4:

                draw\_on\_canvas(canvas, t\_shape)

                draw\_on\_canvas(canvas, r\_shape)

            elif frieze\_group == 5:

                g\_shape = glide\_reflect(t\_shape, repeat\_length//2, height//2)

                g5\_shape = glide\_reflect(r\_shape, -(repeat\_length//2), height//2)

                draw\_on\_canvas(canvas, t\_shape)

                draw\_on\_canvas(canvas, g\_shape)

                draw\_on\_canvas(canvas, r\_shape)

                draw\_on\_canvas(canvas, g5\_shape)

            elif frieze\_group == 6:

                h\_shape = reflect\_h(t\_shape, height//2)

                draw\_on\_canvas(canvas, t\_shape)

                draw\_on\_canvas(canvas, h\_shape)

            elif frieze\_group == 7:

                h\_shape = reflect\_h(t\_shape, height//2)

                draw\_on\_canvas(canvas, t\_shape)

                draw\_on\_canvas(canvas, r\_shape)

                draw\_on\_canvas(canvas, h\_shape)

                draw\_on\_canvas(canvas, v\_shape)

if \_\_name\_\_ == "\_\_main\_\_":

    canvas.pack()

    root.resizable(width = False, height = False)

    make\_frieze('foot\_vert.txt', 5, 300, 200, 3)

    root.mainloop()

Self-Evaluation

I believe that I have successfully completed all of the exercises. Although I did relatively easily complete the first exercises, I did have a bit of difficulties working with the friezes especially with the height (though very simple) and certain manipulations (spinning sidle in particular).

I’d say my code wouldn’t be that difficult to understand for someone that understands python as I’ve added docstrings to every function I’ve made and I believe added a couple notes to the parts of code that could be a bit harder to comprehend. I have also avoided complex code and tried to keep it as simple as possible whilst making sure the code work effectively.

I do have to say I usually name variables, in the code not the function variables, by the first letter of what is should produce, e.g.: rotation 🡪 r

That aspect might make it harder to understand but with the docstrings and the few notes it shouldn’t be relatively hard to understand or deduce what each variable does.

By that logic and despite the one letter variables, it shouldn’t be too hard to modify the code if one were to change it to match their required task.