Assignment 2

COMP9021, Trimester 3, 2025

1 General matters

1.1 Aim

The purpose of the assignment is to:

- develop object-oriented Python programs with proper exception handling;
- parse and analyse combinatorial structures;
- generate TikZ/LaTeX diagrams programmatically;
- handle both small and complex structures efficiently.

1.2 Submission

Your program should be stored in a file named `arches.py`, optionally together with additional files. After developing and testing your program, upload it via Ed (unless you worked directly in Ed). Assignments can be submitted multiple times; only the last submission will be graded. Your assignment is due on November 24 at 11:59am.

1.3 Assessment

The assignment is worth 13 marks and will be tested against multiple inputs. For each test, the automarking script allows your program to run for 30 seconds.

Assignments may be submitted up to 5 days after the deadline. The maximum mark decreases by 5% for each full late day, up to a maximum of five days. For example, if students A and B submit assignments originally worth 12 and 11 marks, respectively, two days late (i.e., more than 24 hours but no more than 48 hours late), the maximum mark obtainable is 11.7. Therefore, A receives $\min(11.7, 12) = 11.7$ and B receives $\min(11.7, 11) = 11$.

Your program will generate a number of `.tex` files. These can be given as arguments to `pdflatex` to produce PDF files. Only the `.tex` files will be used to assess your work, but generating the PDFs should still give you a sense of satisfaction. The outputs of your programs must exactly match the expected outputs. **You are required to use the `diff` command to identity any differences between the `.tex` files generated by your program and the provided reference `.tex` files. You are responsible for any failed tests resulting from formatting discrepancies that `diff` would have detected.**

1.4 Reminder on plagiarism policy

You are encouraged to discuss strategies for solving the assignment with others; however, discussions must focus on algorithms, not code. You must implement your solution independently. Submissions are routinely scanned for similarities that arise from copying, modifying others' work, or collaborating too closely on a single implementation. Severe penalties apply.

2 Open Meanders

2.1 Background

An *open meander* is a combinatorial structure represented as a non-self-intersecting curve that crosses a horizontal line of points, forming arches above and below the line. They can be described by permutations with specific constraints.

Formally, let $(a_1, a_2, \ldots, a_n)$ be a permutation of $\{1, \ldots, n\}$ with $n \geq 2$. Each integer corresponds to a distinct point on a fixed horizontal line. The permutation defines a sequence of arches as follows:

- The first arch is an upper arch, drawn above the line.
- Subsequent arches alternate between upper and lower positions, forming a valid open meander.
- Each arch connects two consecutive points a_i and a_{i+1} in the permutation. The orientation of each arch depends on the relative order of these points:
  - An arch is drawn from left to right if $a_i < a_{i+1}$.
  - An arch is drawn from right to left if $a_i > a_{i+1}$.
- Arches on the same side do not intersect.

The collection of upper and lower arches can be represented symbolically using *extended Dyck words*—one for each side of the line:

- `(` corresponds to the left endpoint of an arch.
- `)` corresponds to the right endpoint of an arch.
- `1` represents an end of the curve (a free endpoint) that lies on that side.

The position of the endpoints depends on the parity of n:

- For even n, both ends of the curve lie below the line.
- For odd n, one end lies above and the other below the line.

Each extended Dyck word therefore encodes the complete structure of the arches on its respective side, though only together do the two sides represent the full open meander.

2.2 Examples

For a first example, consider the permutation $(2, 3, 1, 4)$ and the corresponding generated diagram open_meanders_1.pdf.

- Upper arches extended Dyck word: `(())`
- Lower arches extended Dyck word: `(1)1`

For a second example, consider the permutation $(1, 10, 9, 4, 3, 2, 5, 8, 7, 6)$ and the corresponding generated diagram open_meanders_2.pdf.

- Upper arches extended Dyck word: `(()((()))`
- Lower arches extended Dyck word: `1(())1()()`

For a third example, consider the permutation $(5, 4, 3, 2, 6, 1, 7, 8, 13, 9, 10, 11, 12)$ and the corresponding generated diagram open_meanders_3.pdf.

- Upper arches extended Dyck word: `(()())()(()1)`
- Lower arches extended Dyck word: `(((()1))(()())`

2.3 Requirements

Implement in `arches.py` a class `OpenMeanderError(Exception)` and a class `OpenMeander`.

Objects of type `OpenMeander` are created with `OpenMeander(a_1, a_2, ..., a_n)`, where the arguments form a permutation of $\{1, ..., n\}$ for some $n \geq 2$. You may assume that all arguments are integers.

- If the arguments do not form a permutation of $\{1, ..., n\}$ for some $n \geq 2$, raise

  `OpenMeanderError('Not a permutation of 1, ..., n for some n ≥ 2').`

- If they do not define a valid open meander, raise

  `OpenMeanderError('Does not define an open meander').`

Implement in `OpenMeander` three attributes:

- `extended_dyck_word_for_upper_arches`, a string representing the upper arches;
- `extended_dyck_word_for_lower_arches`, a string representing the lower arches;
- `draw(filename, scale=1)`, a method that generates a TikZ/LaTeX file drawing the open meander.

No error checking is required in the implementation of `draw(filename, scale=1)`. You may assume that `filename` is a valid string specifying a writable file name, and that `scale` is an integer or floating-point number (typically chosen so that the resulting picture fits on a page).

An example interaction is shown in open_meanders.pdf.

Carefully study the three example `.tex` files. Note that the horizontal baseline extends one unit beyond each end of the curve. Also note that the scale common to `x` and `y`, as well as the values for `radius`, are displayed as floating-point numbers with a single digit after the decimal point. The length of the ends of strings is computed as half of the scale of `x` and `y`, and is also displayed as a floating-point number with a single digit after the decimal point.

3 Dyck Words and Arch Diagrams

3.1 Background

A *Dyck word* is a balanced string of parentheses representing a system of nested arches above a horizontal line. The depth of an arch is the number of arches it is nested within, providing a way to analyse the hierarchical structure.

For example, the Dyck word `(()(()(())))` contains arches of depth 0, 1, 2, and 3. Dyck words can be visualised as arches drawn above a horizontal line, with nesting reflected in the vertical stacking of arches.

Unlike open meanders, Dyck words involve only one side of arches (above the line) and do not include endpoints represented by 1. They provide a simplified context for studying nesting depth and arch diagrams, and arches can optionally be visually distinguished by color according to their depth.

When colouring is applied, the following sequence is used: `Red`, `Orange`, `Goldenrod`, `Yellow`, `LimeGreen`, `Green`, `Cyan`, `SkyBlue`, `Blue`, `Purple`. If the maximum depth exceeds 9, the sequence wraps around. For example, depth 10 would use `Red` again, depth 11 `Orange`, etc.

3.2 Examples

For a first example, consider the Dyck word `((((((((((((()))))))))))))` and the corresponding generated diagrams, drawn_dyck_word_1.pdf and coloured_dyck_word_1.pdf.

- There is 1 arch of depth 0.
- There is 1 arch of depth 1.
- There is 1 arch of depth 2.
- There is 1 arch of depth 3.
- There is 1 arch of depth 4.
- There is 1 arch of depth 5.
- There is 1 arch of depth 6.
- There is 1 arch of depth 7.
- There is 1 arch of depth 8.
- There is 1 arch of depth 9.
- There is 1 arch of depth 10.
- There is 1 arch of depth 11.
- There is 1 arch of depth 12.
- There is 1 arch of depth 13.

For a second example, consider the Dyck word `(()(()(())))` and the corresponding generated diagrams, drawn_dyck_word_2.pdf and coloured_dyck_word_2.pdf.

- There are 3 arches of depth 0.
- There is 1 arch of depth 1.
- There is 1 arch of depth 2.
- There is 1 arch of depth 3.

For a third example, consider the Dyck word `((()())(()(()())))` and the corresponding generated diagrams, drawn_dyck_word_3.pdf and coloured_dyck_word_3.pdf.

- There are 5 arches of depth 0.
- There are 2 arches of depth 1.
- There is 1 arch of depth 2.
- There is 1 arch of depth 3.

For a fourth example, consider the Dyck word `(((()(()))(()(()(()))))(((()()))()(()()))))` and the corresponding generated diagrams, drawn_dyck_word_4.pdf and coloured_dyck_word_4.pdf.

- There are 11 arches of depth 0.
- There are 4 arches of depth 1.
- There are 2 arches of depth 2.
- There is 1 arch of depth 3.
- There is 1 arch of depth 4.
- There is 1 arch of depth 5.

3.3 Requirements

Implement in `arches.py` a class `DyckWordError(Exception)` and a class `DyckWord`.

Objects of type `DyckWord` are created with `DyckWord(s)`, where the argument `s` is a nonempty string of parentheses. You may assume that the argument is a string.

- If the argument is the empty string, raise

  `DyckWordError('Expression should not be empty')`.

- Otherwise, if the argument contains characters other than parentheses, raise

  `DyckWordError("Expression can only contain '(' and ')'")`.

- Otherwise, if the string is not balanced, raise

  `DyckWordError('Unbalanced parentheses in expression')`.

Implement in `DyckWord` three attributes:

- `report_on_depths()`, a method that outputs the number of arches at each depth, ordered from smallest to largest depth;
- `draw_arches(filename, scale=1)`, a method that generates a TikZ/LaTeX file drawing the arches;
- `colour_arches(filename, scale=1)`, a method that generates a TikZ/LaTeX file drawing the arches coloured according to their depth.

No error checking is required in the implementation of both methods. You may assume that `filename` is a valid string specifying a writable file name, and that `scale` is an integer or floating-point number (typically chosen so that the resulting picture fits on a page).

An example interaction is shown in dyck_words.pdf.

Carefully study the eight example `.tex` files (four for drawing arches, four for colouring arches). Note that the horizontal baseline extends one unit beyond the leftmost and rightmost arches. Note that the scale common to `x` and `y` is displayed as a floating-point number with a single digit after the decimal point. Arches are drawn from the **leftmost left end** to the **rightmost left end**. Arches are coloured from **largest depth to smallest depth**, and for arches of the same depth, **from leftmost left end to rightmost left end**.