

Scenario 1: Permutation Generator Proposal

Team 35: Aaryaman Sharma, Eloise Vriesman, Kin Cheung, Pooja Chhaya

1. Introduction.....(Page 2)
2. Framework Used for Implementation.....(Page 2)
3. List of Components.....(Page 2)
 - a. Component 1 (Page 2)
 - b. Component 2 (Page 5)
 - c. Component 1 & Component 2 Interaction (Page 6)
4. User Interface & Example Exercises(Page 7)

1. Introduction:

This proposal outlines the teaching tool we plan to make on the topic of permutations. It is a mathematics exercise tool designed to help first year undergraduate students studying computer science practice a range of questions on permutations. This is done by asking similar questions as would be seen in a discrete mathematics module.

The program, titled “Permutation Generator”, randomly generates a permutation in Cauchy’s two-line notation. The user can pick the cardinality of the set, ranging from 2-12 elements, and the “Permutation Generator” then asks students to solve three problems pertaining to the permutation.

The three parts of the question are: calculating the sign of the permutation, the order of the permutation, and decomposing the permutation into disjoint cycles. The program marks the questions and displays which sections the student was able to successfully answer. If any answers were discovered by the program to be incorrect, the correct answer is displayed. The user will then be able to move onto the next question

The game aims to provide Computer Science and Mathematics students a tool to understand the concept of permutations and how to mechanically solve problems relating to the topic.

Permutations and their relevance in the general field of discrete mathematics are crucial because they teach a method of problem solving and logical thinking that is required in the field of Computer Science. By using the “Permutation Generator” teaching tool, students will be able to develop intuition regarding permutations. This will allow them to build up their confidence and knowledge of discrete mathematics, enabling computer science students to apply the skills they have learnt from our tool to many computational problems. These include applications such as counting and sorting algorithms, as well as data validation.

2. Framework

The program will be deployed as a web app. Django (based on python) will be used due to its versatility, ability to integrate the front-end and back-end seamlessly using HTML templates, and thorough documentation, which makes development more straightforward.

The service used to deploy the website on will most likely be Heroku. This is due to the fact it has a competent free tier and integrates well with Django.

3. List of Components

There are 2 main components involved in the design of the web app.

a. Component 1

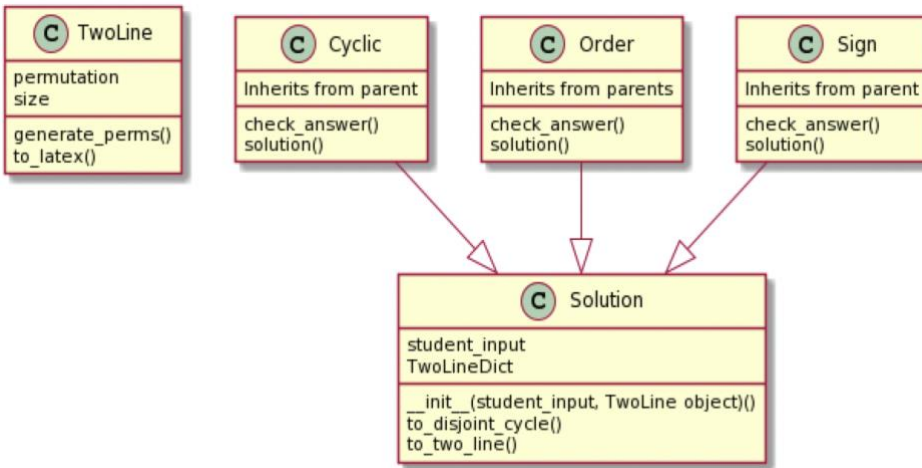


Figure 1: A class diagram to generate and store permutations

The first component deals with creating classes to generate and store permutations, finding solutions to the three problems, and comparing them to the student's input. This is shown in Figure 1. These classes are described in more detail below:

- **TwoLine Class:** Deals with generating permutation and encoding it as a dictionary
 - Parameters:
 - Size
 - Optional - Permutation
 - Attributes (Created during object initialization):
 - self.size = Size
 - self.permutation = generate_perms() or Permutation parameter if provided
 - Methods:
 - generate_perms()
 - to_latex()

Method: generate_perms()

This method generates a set of *integers* (containing all integers 1 to *size*) with a user defined *size*. It randomly shuffles this set to generate a random permutation of the elements in the set. The information of the set and its permutation is stored in the form of a *dictionary* defined as shown in Figure 2.

Name	Type	Description
element	integer	The key of the dictionary. Stores the elements of the set
map	integer	Stores the elements' corresponding mapping in the generated permutation

Figure 2: A table defining the dictionary storing the set & its permutation

When a new TwoLine object is created, the self.permutation attribute calls this method and sets its value to the generated dictionary.

Method: to_latex()

This method simply converts the dictionary of permutation to a parentheses matrix latex form so it can be displayed on a webpage.

- **Class: Solution** - Parent class of specific solution classes
 - Parameters:
 - TwoLine object
 - student_input
 - Attributes:
 - self.student_input = student_input
 - self.twoLineDict = self.permutation attribute of TwoLine object
 - Methods:
 - to_disjoint_cycles()
 - to_two_line()

Method: to_disjoint_cycle()

This method converts the dictionary format of the permutation into cyclic form. The cycles are represented as lists of numbers, with the order of numbers being in the canonical cyclic form. This method returns a list of all the cycle sublists. Since the cycles are disjoint, the order in which we return these sublists doesn't matter.

Method: to_two_line()

Given a permutation in cycle notation as described above, this method converts it back to a dictionary of permutations. Double for-loops are used to iterate through the list of lists, with the key being the current iteration in the inner for-loop, and the value being the next element in the iteration. At the last element in the inner for-loop, the value will be the first element in the sub list since that is how cycle notation is defined.

These methods gain their importance as they form the basis of solving the problems we give to the user. Each subclass of *Solution* inherits these methods to help them find the solution to the corresponding permutation and compare it to the user input. These subclasses are as follows:

- **Subclass: Order**
 - Attributes: Inherited from parent class *Solution*
 - Methods:
 - solution()
 - check_answer()
- **Subclass: Sign**
 - Attributes: Inherited from parent class *Solution*

- Methods:
 - solution()
 - check_answer()
- **Subclass: Cyclic**
 - Attributes: Inherited from parent class *Solution*
 - Methods:
 - solution()
 - check_answer()

Each subclass deals with one of the three problems the students must answer. The solution() method in each class takes a permutation in dictionary format and finds the solution for a specific problem relating to the permutation.

For example, the solution() method in the *Cyclic* subclass simply calls the to_disjoint_cycle() method from the parent class, and changes it to a string to make it look like the traditional cycle notation. Similarly, the solution() method in the *Order* subclass also calls the to_disjoint_cycle() method, but instead finds the lowest common multiple of the lengths of the cycles, to find the order of the permutation. The other classes have similar methods of finding the solution to a specific problem relating to the given permutation.

The check_answer() method in all of the subclasses simply compares the student input with the solution returned by their solution() method, returning True if they are equal and False if they are not. However, in the case of cycle notation, there can be multiple ways of representing the same permutation. Thus, the check_answer() method actually calls the to_two_line() method on the student_input to convert it to a permutation in dictionary format and checks if it's equal to the original permutation.

b. Component 2

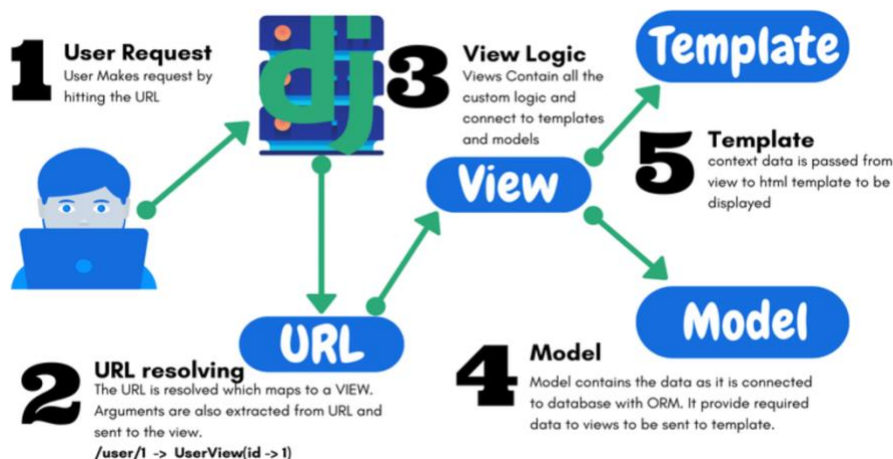


Figure 3: A flowchart displaying the interaction between systems

Neumann, I., 2021. A Non-Scary Introduction to Django. [online] Cup of Code. Available at: <<https://cupofcode.blog/intro-to-django/>> [Accessed 9 February 2022].

The second component of our web app is the website. The tool will use the Django Model-View-Template (MVT). Figure 3 shows a basic flowchart of how these systems interact.

The *Model* stores fields and attributes about any data on the website we want to store and manipulate. For example, a form on the website where the user chooses the size of the permutation can be stored as a *SetSize* model. This model could have the field *size* that stores the value of whatever size the user selected. A *StorePermutation* model could be used to save the permutation that is generated whenever the user generates a new one. An *Answer* model could store the answers the user enters, with different fields for the type of question they answered.

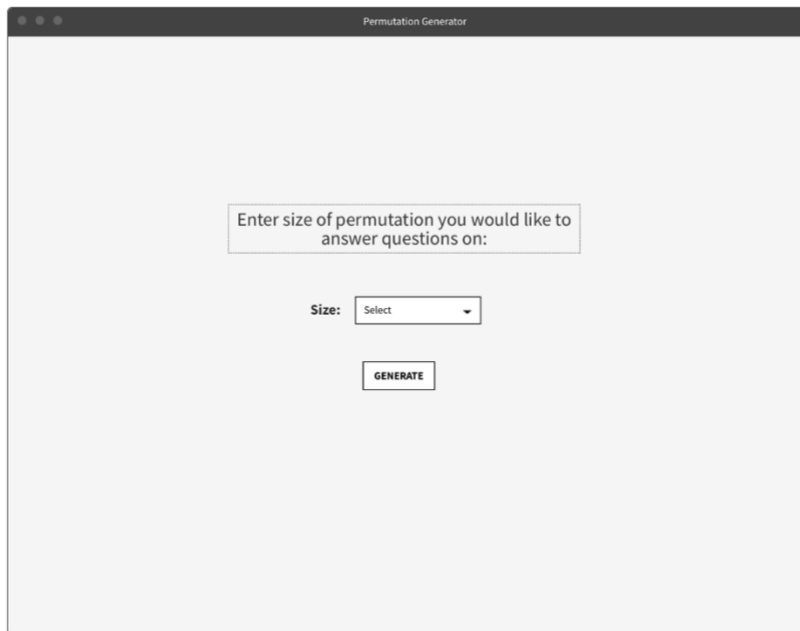
The *View* deals with receiving web requests and passing on web responses. While sending requests, it can also send variables alongside it for the *Template* to process. For instance, it can integrate with our *Answer* model to automatically generate user-input forms based on the fields in our model. It can then send those forms as variables to the *Template* so they can be displayed on the website. It can also, for example, store and retrieve the permutation in the *StorePermutation* model and save it to an external database. This could be used for maintaining the permutation displayed on screen even when the website refreshes. When receiving responses, such as user answers from a form, you can specify custom methods to parse the data, and then pass it on to the *Solution* subclasses in component 1. It can then receive whether the answers were correct and send this information to the *Template* to show to the user.

The *Template* holds the HTML layout of the website, specifying how the page should be displayed. External libraries such as Bootstrap and MathJax can be imported, like most HTML files, to improve the function of the website with mathematical formulas, as well as to improve the website's simplicity and design. However, unlike a normal HTML file, it can send and receive variables from the *View* and use common python functions on those variables. For example, the *View* might send a dictionary variable specifying what answers the users got wrong and the solutions to those questions. The *Template* can use a for loop to iterate through the answers and integrate it in the HTML file to show on the website.

c. Interaction Between Component 1 and Component 2

The project folder will be laid out in the conventional Django format, with the *Model*, *View* and *Templates* as their own files. Other Django specific files and folders will also be included as needed. The classes in component 1 will be stored as a separate file, which can be imported into the *View* file. The *View* can then use the classes in the component 1 file to generate new permutations based on the size the user inputs, validate user answers received from the *Template*, calculate the correct solutions etc. It can then update the *Model* to reflect any data changes and pass on any relevant variables to the *Template* so it can be displayed on the website.

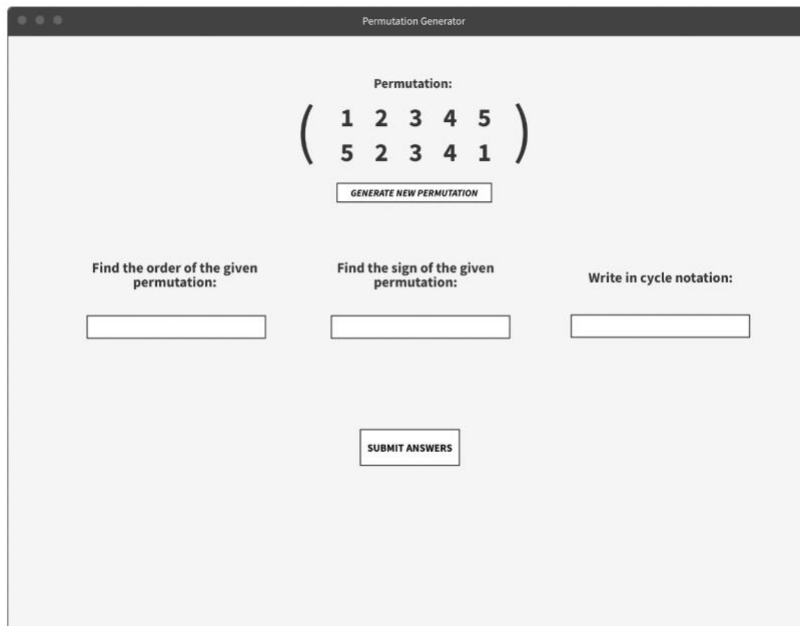
4. User Interface & Example Exercises:



The wireframe shows a window titled "Permutation Generator". Inside, there is a text box with the prompt "Enter size of permutation you would like to answer questions on:". Below this is a label "Size:" followed by a dropdown menu currently showing "Select". At the bottom of this section is a button labeled "GENERATE".

Figure 4: Wireframe of the website's opening page

Figure 4 is a wireframe demonstrating the layout of the website's opening page. The user will be able to input the number of elements they would like the permutation to contain, ranging from 2-12. This allows students to start with simpler questions and increase the difficulty as their confidence and knowledge on permutations increases.



The wireframe shows a window titled "Permutation Generator". At the top, it says "Permutation:" followed by a 2x5 grid of numbers in parentheses: $\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 2 & 3 & 4 & 1 \end{pmatrix}$. Below this is a button labeled "GENERATE NEW PERMUTATION".

Below the permutation, there are three questions, each with a text input field:

- Find the order of the given permutation:
- Find the sign of the given permutation:
- Write in cycle notation:

At the bottom of the page is a button labeled "SUBMIT ANSWERS".

Figure 5: Wireframe of the question page

Figure 5 displays the layout of the question page of the Permutation Generator website. A random permutation is generated based on the size inputted by the student previously. The user will then calculate and enter their values for the permutation's order, sign and the cycle notation. Once all three fields have been filled in the student is able to submit their answers and receive feedback on which questions were answered correctly, as demonstrated by figure 6. If any questions were incorrectly answered, the website displays the correct answer, and the student will be able to move onto the next randomly generated question.

Permutation Generator

Permutation:

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 2 & 3 & 4 & 1 \end{pmatrix}$$

Find the order of the given permutation:

INCORRECT

CORRECT ANSWER: 2

Find the sign of the given permutation:

CORRECT

Write in cycle notation:

CORRECT

NEXT QUESTION

Figure 6: Wireframe of the question page