**Acknowledgement**

First and foremost, I would like to thank our project guide Mrs. Suchita Mandhare who guided us in doing these projects. He provided us with invaluable advice and helped us in difficult periods. His motivation and help contributed tremendously to the successful completion of the project.

Besides, we would like to thank all the teachers who helped us by giving us advice and providing the equipment which we needed.

Finally, we would like to thank my friends, classmates and my family who helped and motivated us to work on this project.

**INDEX**

|  |  |  |
| --- | --- | --- |
| **Sr No** | **Particular** | **Pg No** |
| 1 | Overview | 3 |
| 2 | Description of Existing System | 4 |
| 3 | Limitations of present system | 5 |
| 4 | Proposed System and its advantages | 6 |
| 5 | Technologies Used | 7 |
| 6 | Stakeholders | 8 |
| 7 | Gantt chart | 9 |
| 8 | Event Table | 10 |
| 9 | Use case Diagram | 11 |
| 10 | Entity-Relationship Diagram | 13 |
| 11 | Class Diagram | 14 |
| 12 | Data Flow Diagram | 17 |
| 13 | Sequence Diagram | 18 |
| 15 | State Chart Diagram | 20 |
| 16 | System Coding | 21 |
| 17 | Screen Layouts and Report Layouts | 69 |
| 18 | Future Enhancements | 77 |
| 19 | Bibliography | 78 |

**Overview**

Maths has always been a complex subject for students around the world. Before the dawn of the internet the only source of mathematical problems were books. Since, books were not always available to everyone, it was difficult to improve on one’s mathematical skills. But today in the era of internet, many other sources of knowledge and education are easily accessible.

Although the means have changed the source of these questions remains the same. To counter this, matter the proposed system is an API that is developed to generate unique and customizable questions on demand.

The API will take the following inputs: (1) Types of question (2) Number of Questions. The API will give the following output: json output with the body containing the requested questions.

The goal is to create a platform for students so they can easily practice mathematical problems to improve their skills.

**Description of Existing System**

Some examples of existing systems are [elebetsamer](https://github.com/elebetsamer)/[**math-worksheet-generator**](https://github.com/elebetsamer/math-worksheet-generator), [lukew3](https://github.com/lukew3)/**[mathgenerator](https://github.com/lukew3/mathgenerator)** and, [januschung](https://github.com/januschung)/[**math-worksheet-generator**](https://github.com/januschung/math-worksheet-generator).

The math-worksheet-generator by Elebetsamer is a great example of a simple math question generator. This project is aimed to generate basic math worksheets and was created using an angular framework in Typescript. It can produce four different types of questions: addition, subtraction, multiplication, and division. You can also change the number of addends, subtrahends, factors, and divisors, as well as their values. The application is user-friendly and easy to use. The produced questions are also displayed in a well-organized worksheet that you may download and print.

Lukew3’s mathgenerator is a boundless project written in python2. It's a complete Python library that generates math questions on a variety of topics. The documentation on the repository's github page is simple to follow and library is effortless to install. Math questions ranging from basic arithmetic to calculus, geometry, and statistics are included in the project. The following are some of the types of questions that this project can generate:

1. Addition, Subtraction, Multiplication, Division, Square root, Square, Percentage of number, etc from Basic Algebra section.
2. Power Rule Differentiation, Power Rule Integration, Differentiation, Definite Integral of Quadratic Equations, tec from Calculus section.
3. Binary 1’s Complement, Modulo Division, Decimal to Binary, Fibonacci Series, Binary 2’s complement, etc from Computer Science section.
4. Area of Triangle, Third Angle of Triangle, Pythagorean Theorem, Volume and Surface are of Cylinder, Cuboid, Cone, etc from Geometry section.
5. Combination of objects, Permutation, Probability of a certain sum appearing on face of dice, Mean, Median, Mode, etc, from Statistics section.
6. Least Common Multiple, Greatest Common Divisor, Prime Factorisation, Geometric Progression, Celsius to Fahrenheit, etc from Miscellaneous section.

A few honorable mentions of existing work are [januschung](https://github.com/januschung)/[**math-worksheet-generator**](https://github.com/januschung/math-worksheet-generator), [Teacher’s Corner](https://worksheets.theteacherscorner.net/make-your-own/math-worksheets/basic-math/math.php), Wolramalpha and, [mathsbot.com’s question generator](https://mathsbot.com/questionGenerator).

**Limitations of Existing Systems**

Existing systems like Wolfram|Alpha are very powerful but are sealed behind paywalls and subscriptions. Some other systems like Elebetsamer’s math-worksheet-generator, Lukew3’s mathgenerator and januschung/math-worksheet-generator are free but are also self-hosted meaning you need technical skills to use it.

Limitations for Elebetsamer’s math-worksheet-generator:

* Limited types of questions. That is only four types of questions are available.
* No control over the amount of each type of question on a worksheet.

Limitations for Lukew3’s mathgenerator:

* Cannot generate more than 1 question at once.
* Not user friendly. There is a surplus of options in a single list of available question types.
* Cannot customize generation of question.
* Many questions are for specific use case only.

Many others proprietary like <https://mycbseguide.com/> and [VINZ](https://vizn.co.in/) are user friendly and convenient but the questions are fixed and static, it is fetched from a remote database of questions.

**Proposed System and Its Advantages**

The proposed system is a maths question generator.

Objective: Build an API that can generate maths questions. The api can be queried to obtain maths question on a requested topic. The user must also have some control over the type and level of question that will be generated.

Functionality: The API will take the following inputs: (1) Types of question (2) Number of Questions. The API will give the following output: json output with the body containing the requested questions.

The API will be able to generate the following types of questions:

* Basic operation questions (Addition, Subtraction, Multiplication, Division)
* LCM and HCF
* Linear equations with 2 variables
* Quadratic equations
* Profit Loss Percentage
* Square of Numbers
* Factorial of Numbers
* Nth term in Fibonacci Series
* Permutation and Combination

Advantages:

* The api can be called from the browser using parameters, or via a computer program through code with a json header body. Outputs in xml and json.
* Implemented algorithms for question generation are capable of generating specific to a wide range of questions for a given type.
* Algorithms are well implemented and optimized to generate a large number of questions in all-together.

**Technologies Used**

* Programming Environment-
  + **Operating System**: Windows
  + **Language**: Python 3
  + **Code Editor**: Visual Studio Code
  + **Browser**: Chrome
* Language, Libraries and Frameworks used-
  + **Python Language**: Python is a high-level, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small- and large-scale projects.
  + **Flask**: Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions.
  + **PyQT5**: PyQt5 is a comprehensive set of Python bindings for Qt v5. It is implemented as more than 35 extension modules and enables Python to be used as an alternative application development language to C++ on all supported platforms including iOS and Android.

**Stakeholders**

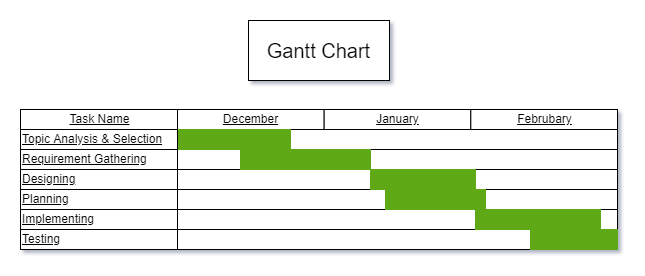
Individuals and organizations who are actively involved in the project, or whose interests may be positively or negatively affected as a result of project execution or successful project completion.

The project's stakeholders are as follows:

* Developer: The developer is the main person in charge of the making the system. All features of the system are well-understood by the developer. The developer is in responsible of keeping track of the information provided by the users.
* End User: The end user is the person who will use this application to benefit from themselves.

**Gantt Chart**

Gantt charts are a type of bar chart that depicts a project's progress. The tasks to be accomplished are shown on the vertical axis, while the time intervals are listed on the horizontal axis. The width of the horizontal bars in the graph shows the time of each action.



**Event Table**

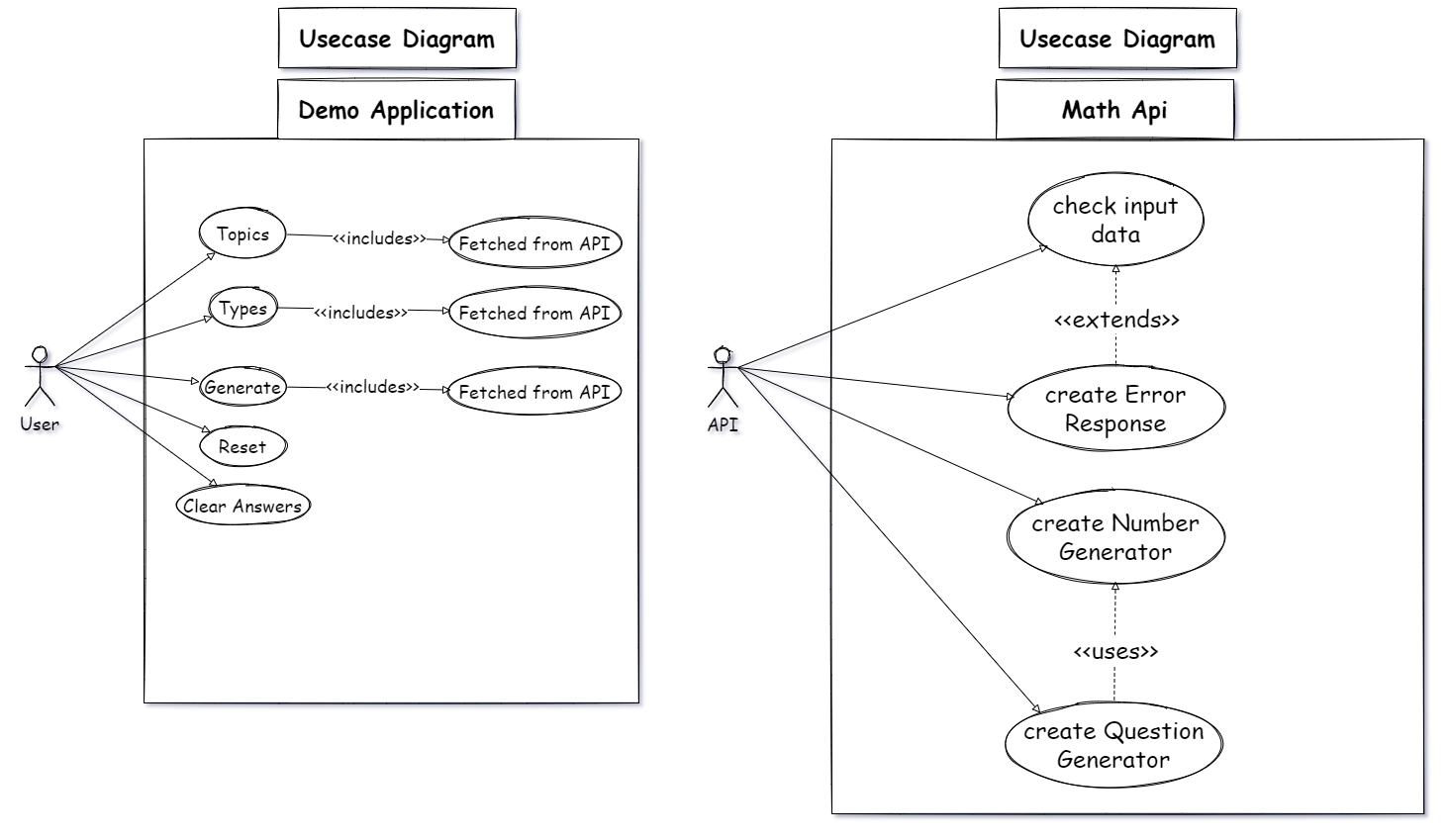
Event Table is a catalogue of use cases listed by event. Contains detailed information

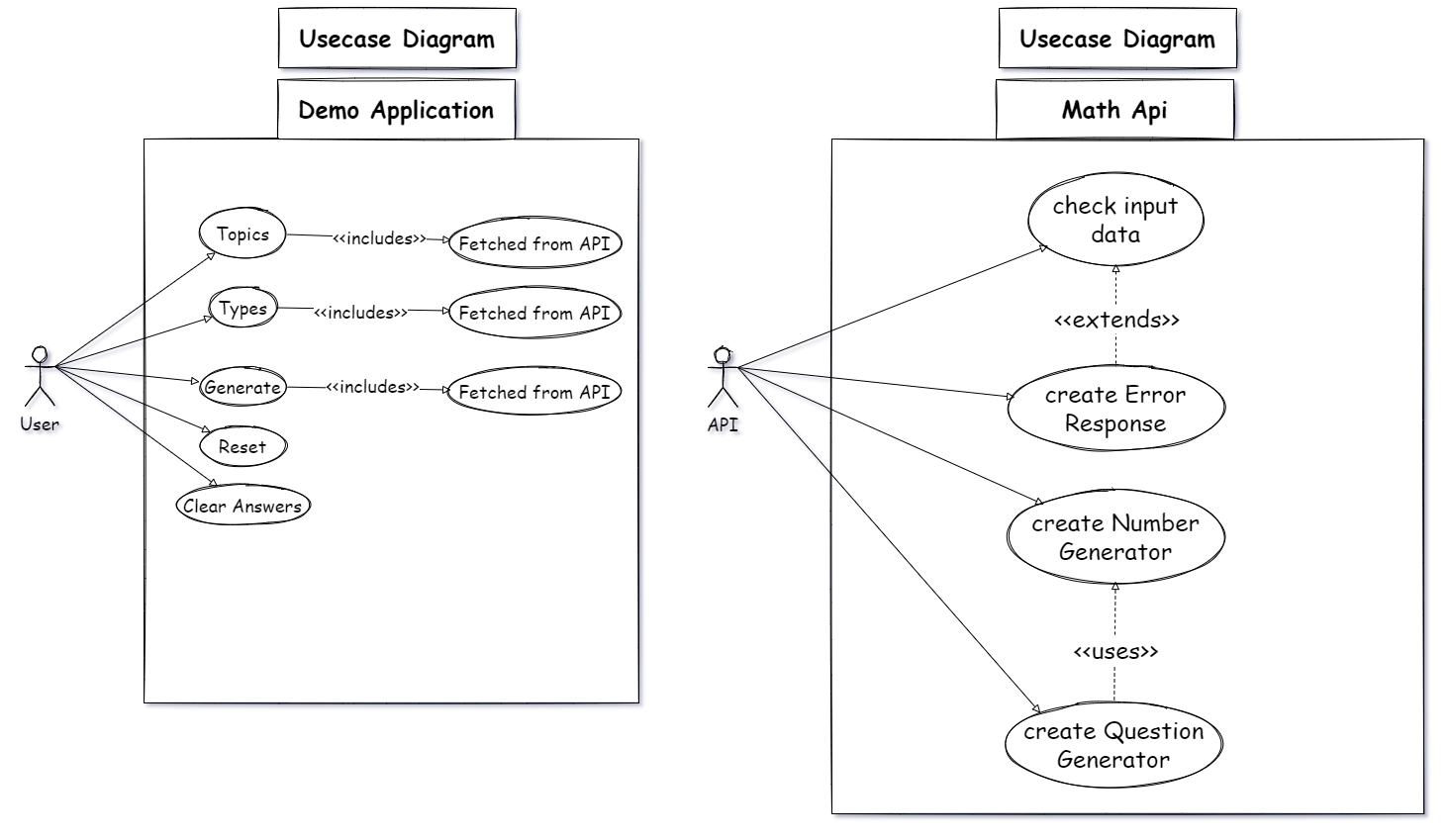
* Trigger: Signal that indicates an event has occurred.
* Source: External agent that initiates event and supplies data for the event.
* Response: Output produced by the system.
* Destination: External agent that receives the response.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Event Table | | | | | |
| Event | Trigger | Source | Use Case | Response | Destination |
| user clicks generate btn | topic, type, no of questions, lower limit, upper limit | User | Fetch questions from api | Questions | User |
| user clicks reset btn | input fields | User | Reset Input Fields |  | User |
| user clicks clear btn | answer list | User | Clear Answers |  | User |
| user selects topic | list available topics | User | select topic | selected topic |  |
| user selects type | list available types | User | select type | selected type |  |
| api checks input data | input data | API | Verifies Input Data | None/Exception | API |
| api requests a number generator | lower limit, upper limit | API | Creates Number Generator | Number Generator | API |
| api requests a question generator | topic, type | API | Creates Question Generator | Question Generator | API |
| api requests a error response | Exception | API | Creates Error Response Object | Response Object | API |

**Use Case Diagram**

Use Case Diagrams are used to summarize the details of any system's users (also known as actors) and their interactions with the system.





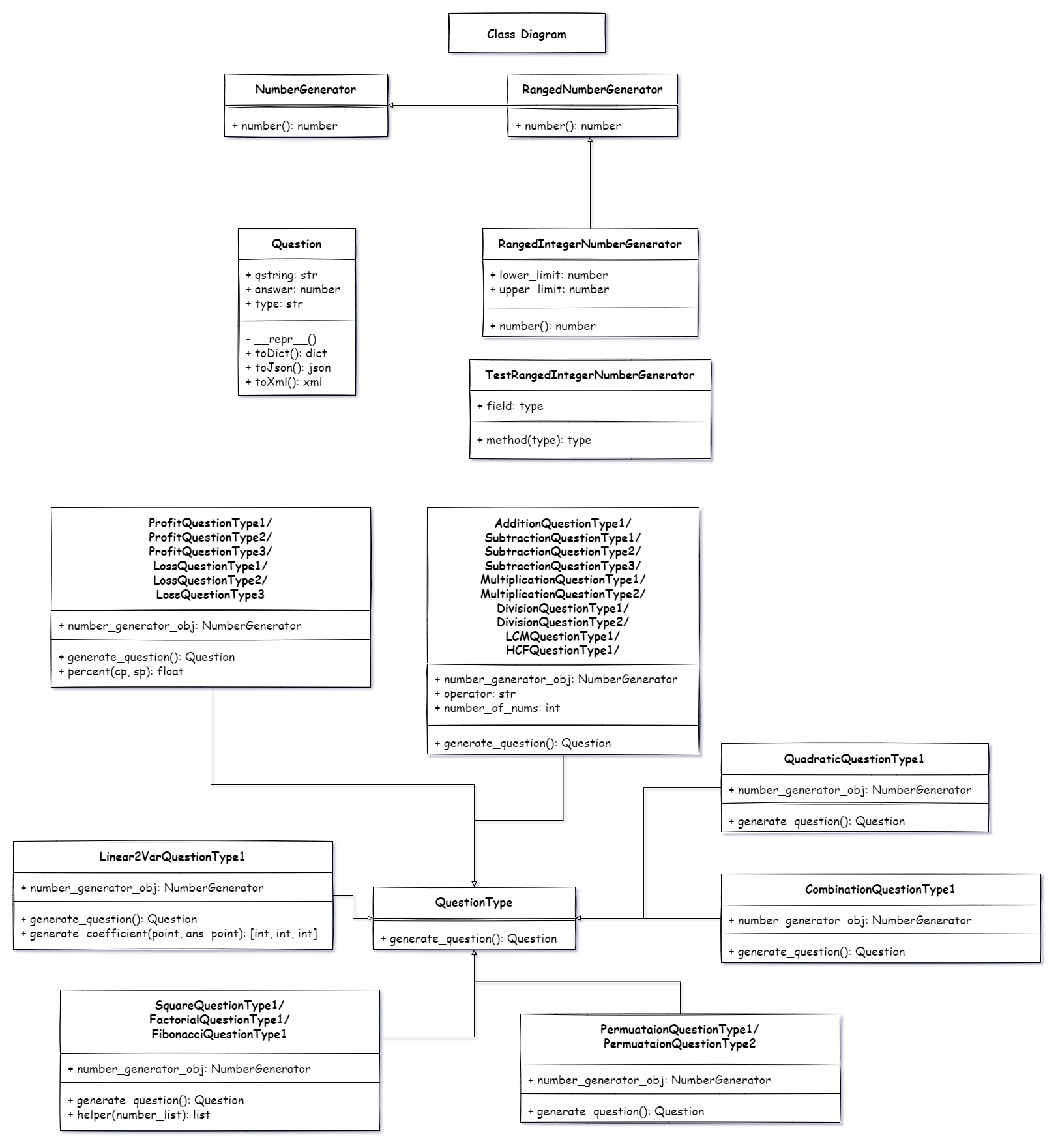
**Entity-Relationship Diagram**

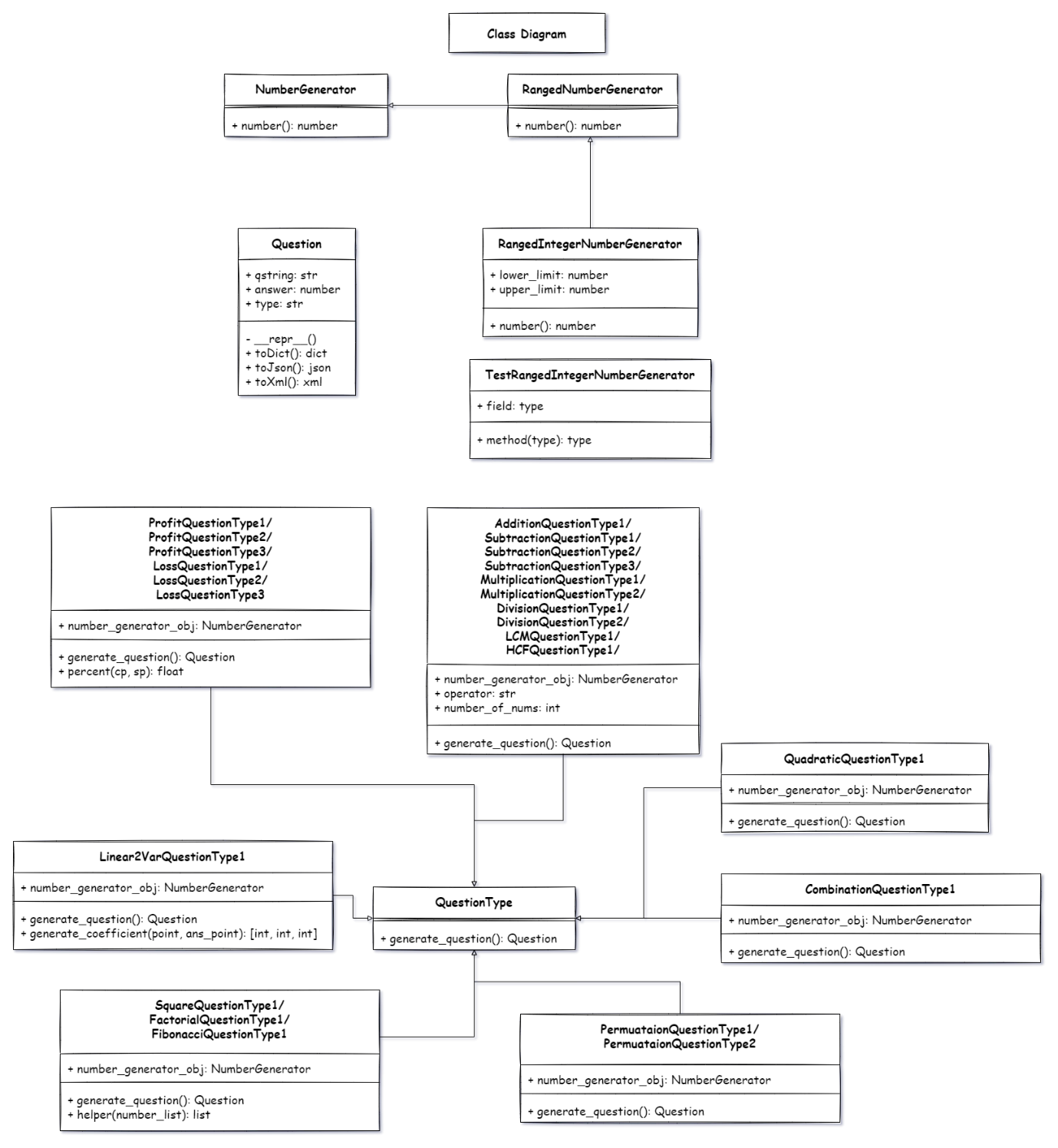
Entity Relationship Diagram (ERD) shows the relationships of entity sets stored in a database. An entity in this context is an object, a component of data. An entity set is a collection of similar entities. These entities can have attributes that define its properties.

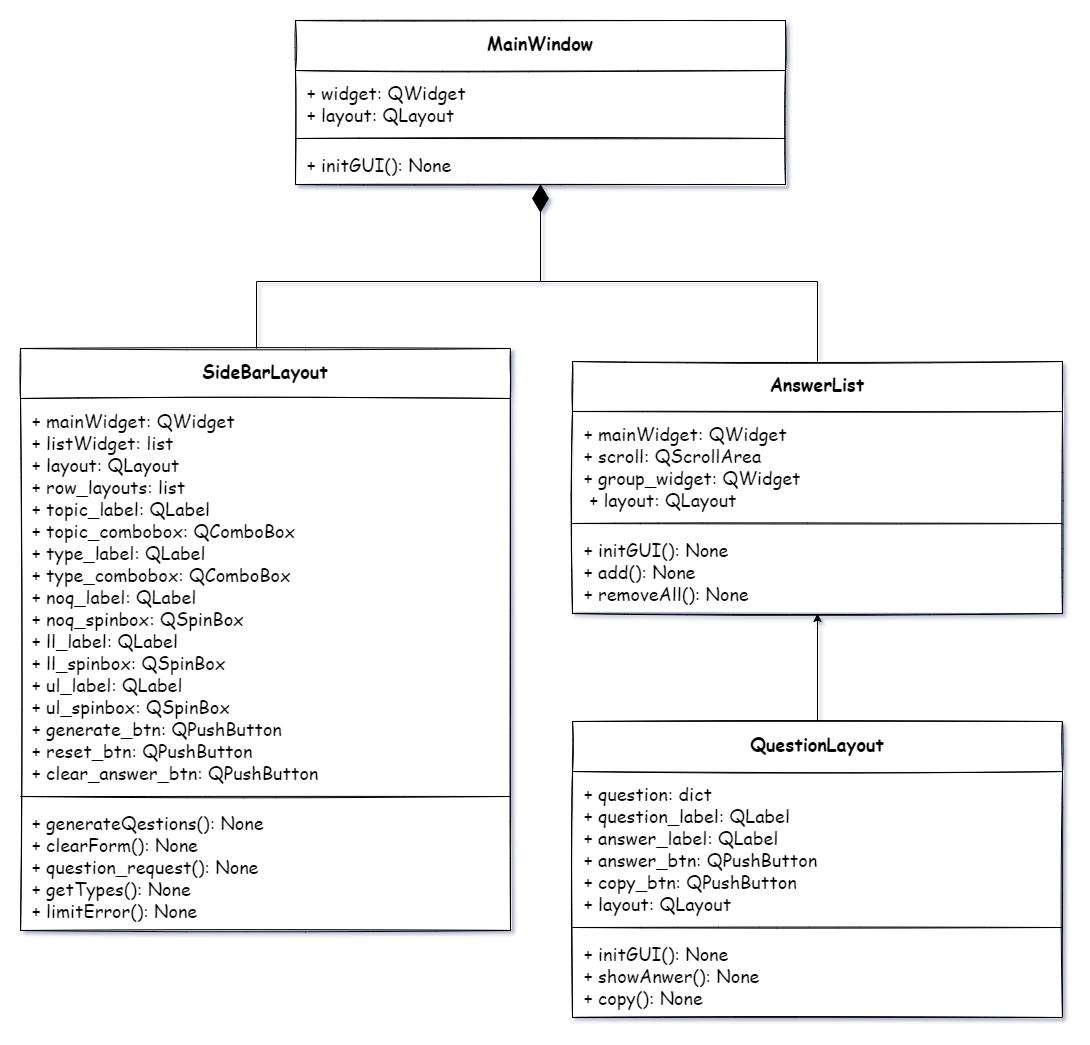


**Class Diagram**

Class diagram is a static diagram. It represents the static view of an application. Class diagram is not only used for understanding and documenting different aspects of a system but also for constructing executable code of the software application. Class diagram describes the attributes and operations of a class and also the constraints imposed on the system. Class diagram shows a collection of classes, interfaces, associations, collaborations, and constraints. It is also known as a structural diagram.

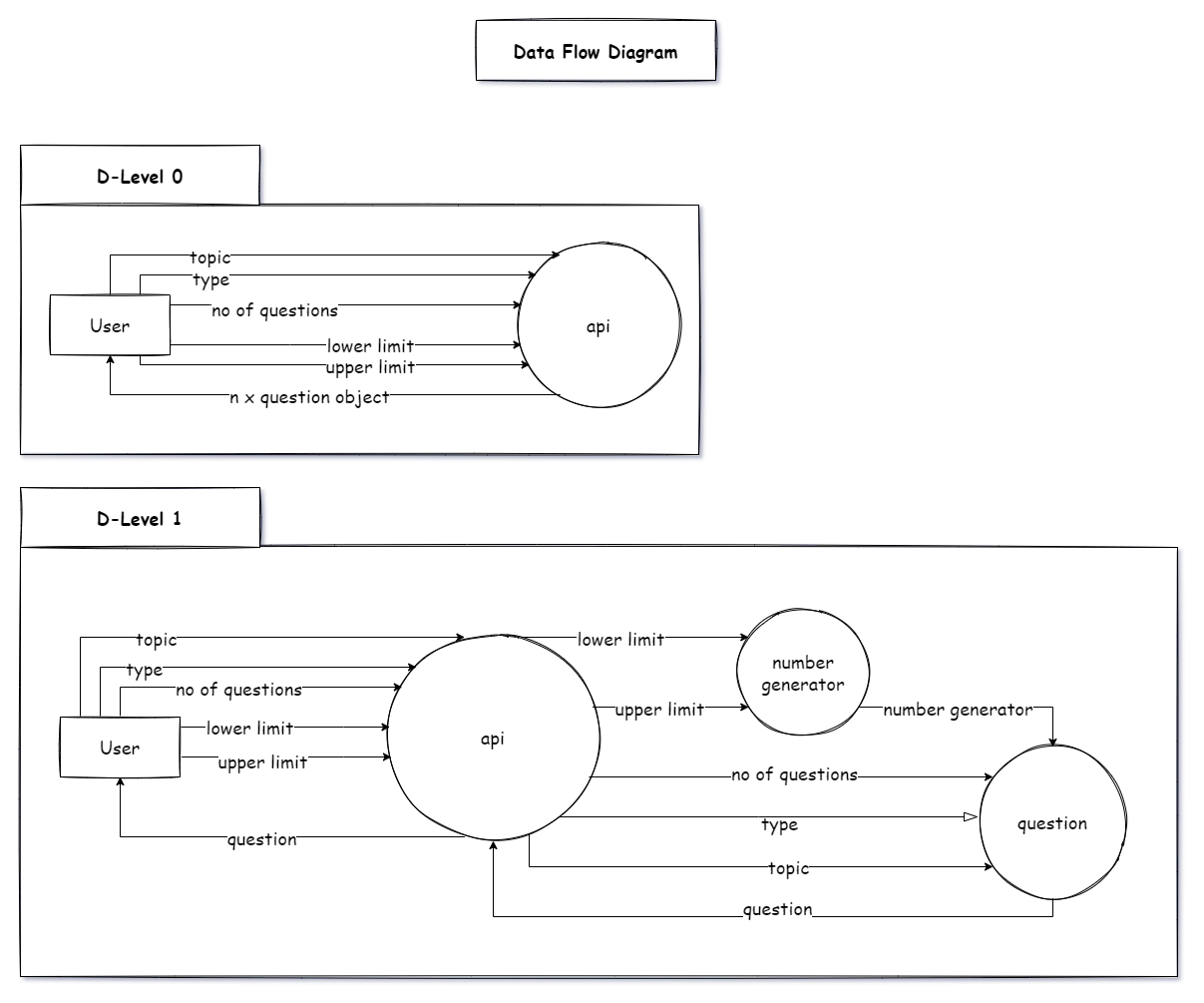


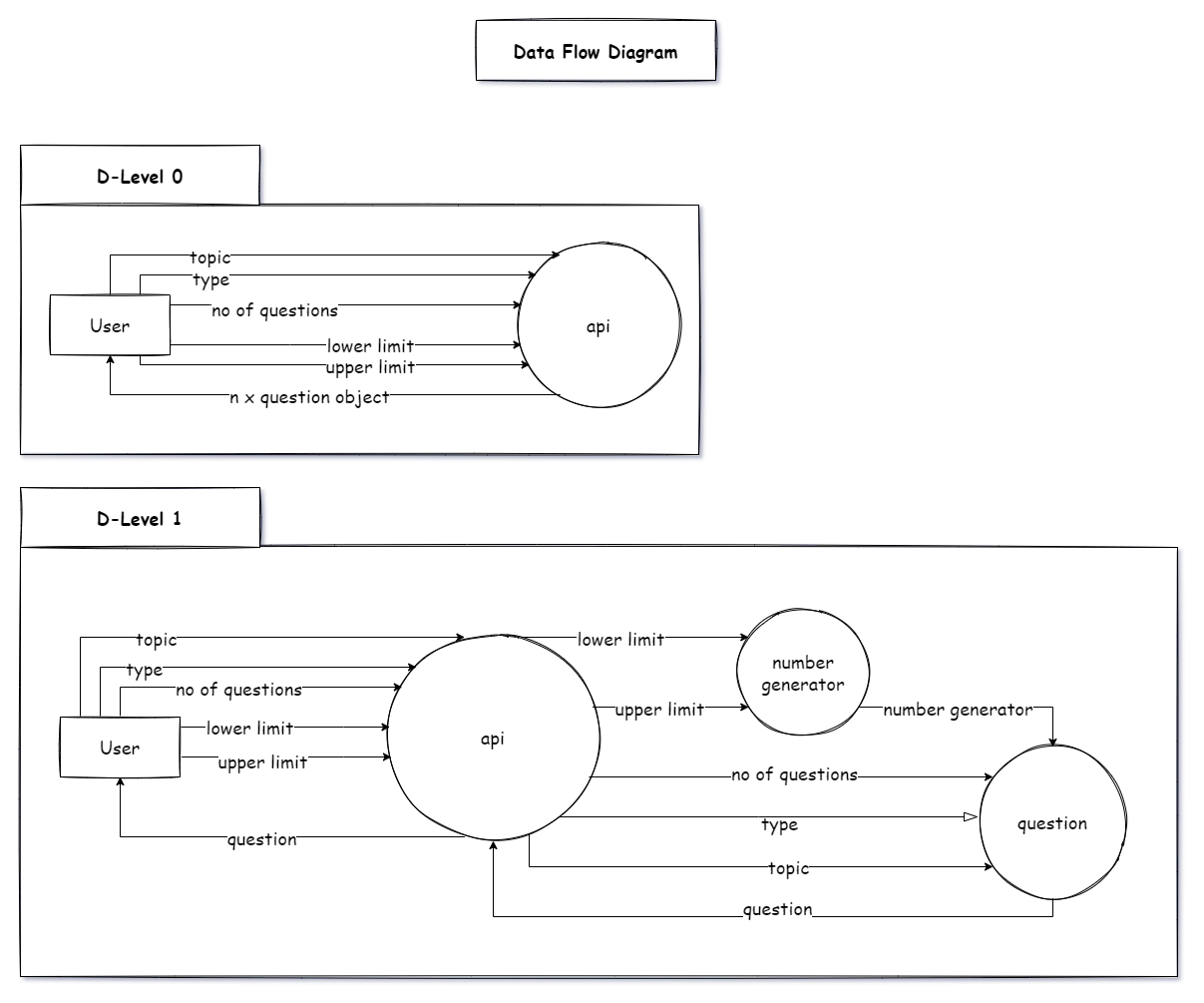


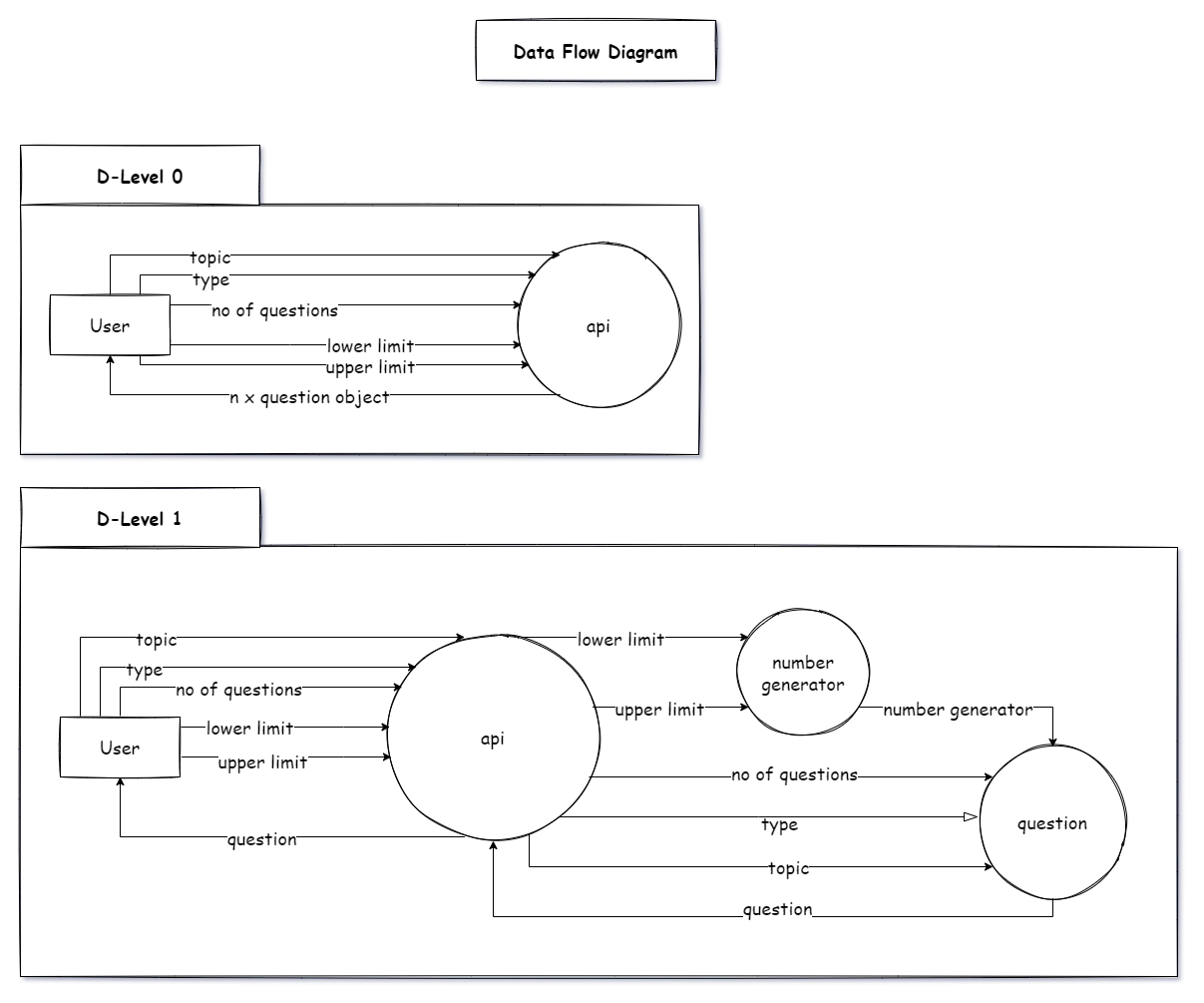


**Data Flow Diagram**

Data Flow Diagram (DFD) describes the in and out the flow of information for any process or system. It uses defined symbols like rectangles, circles and arrows, plus short text labels, to show data inputs, outputs, storage points and the routes between each destination.

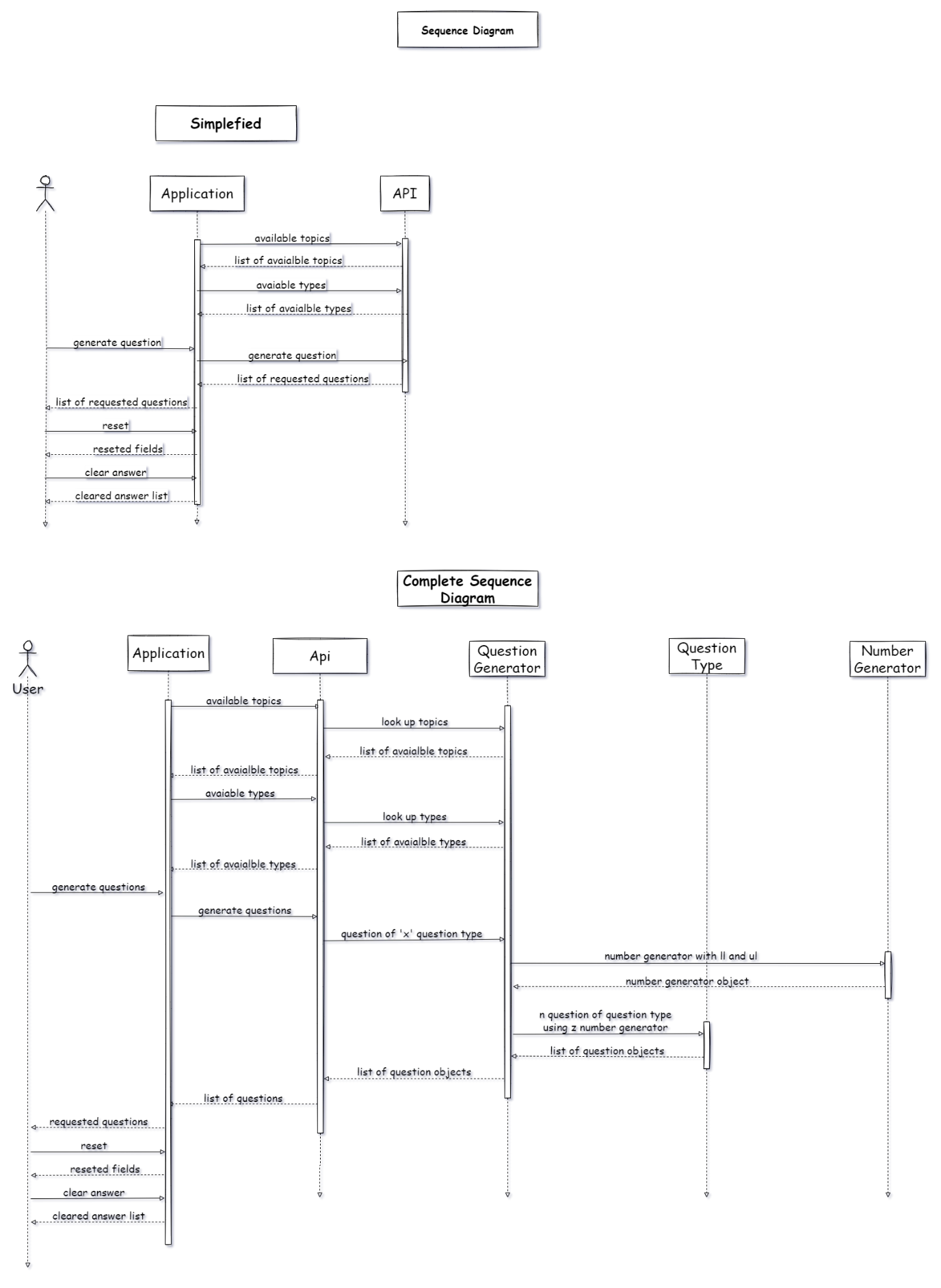


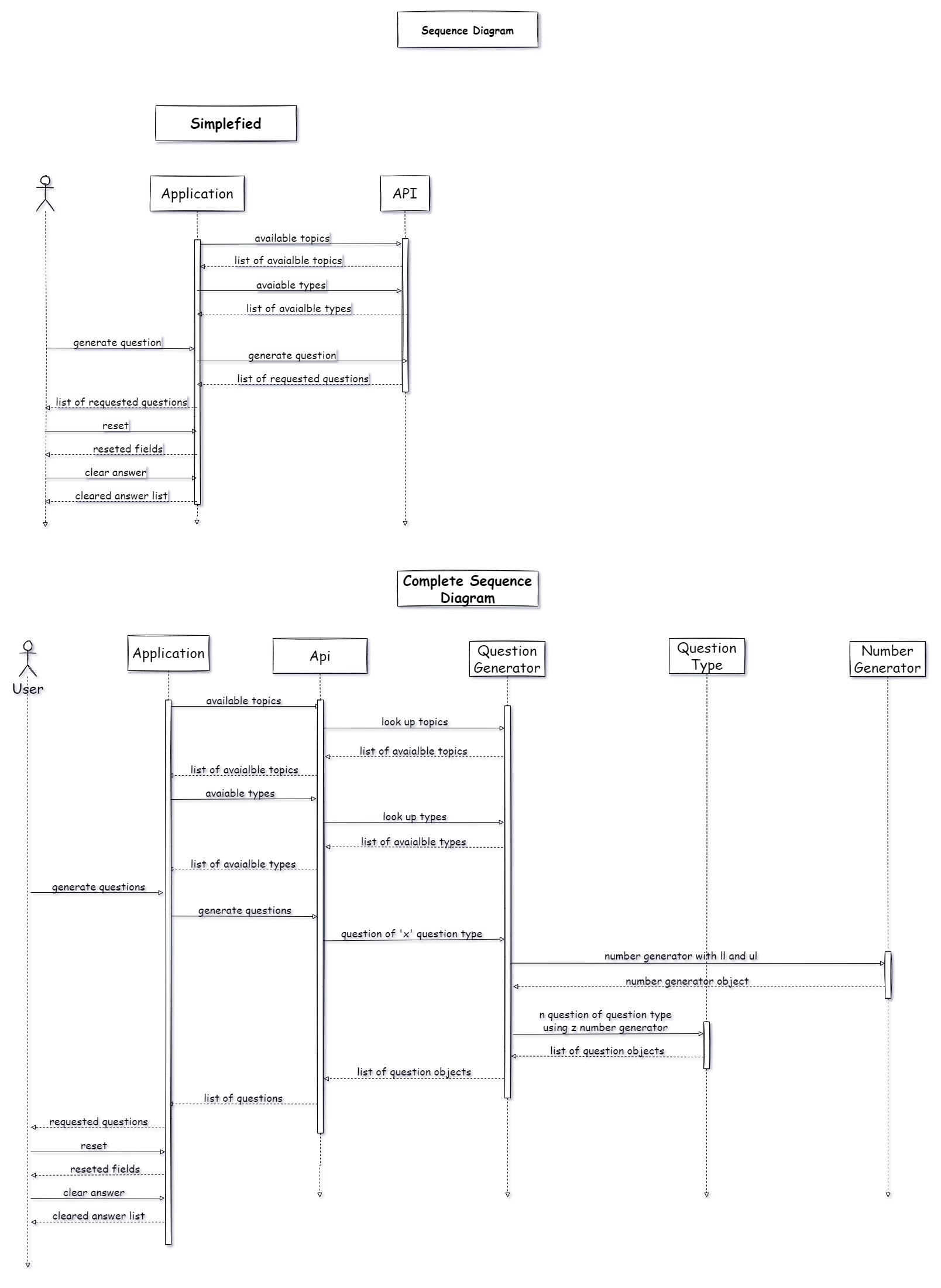


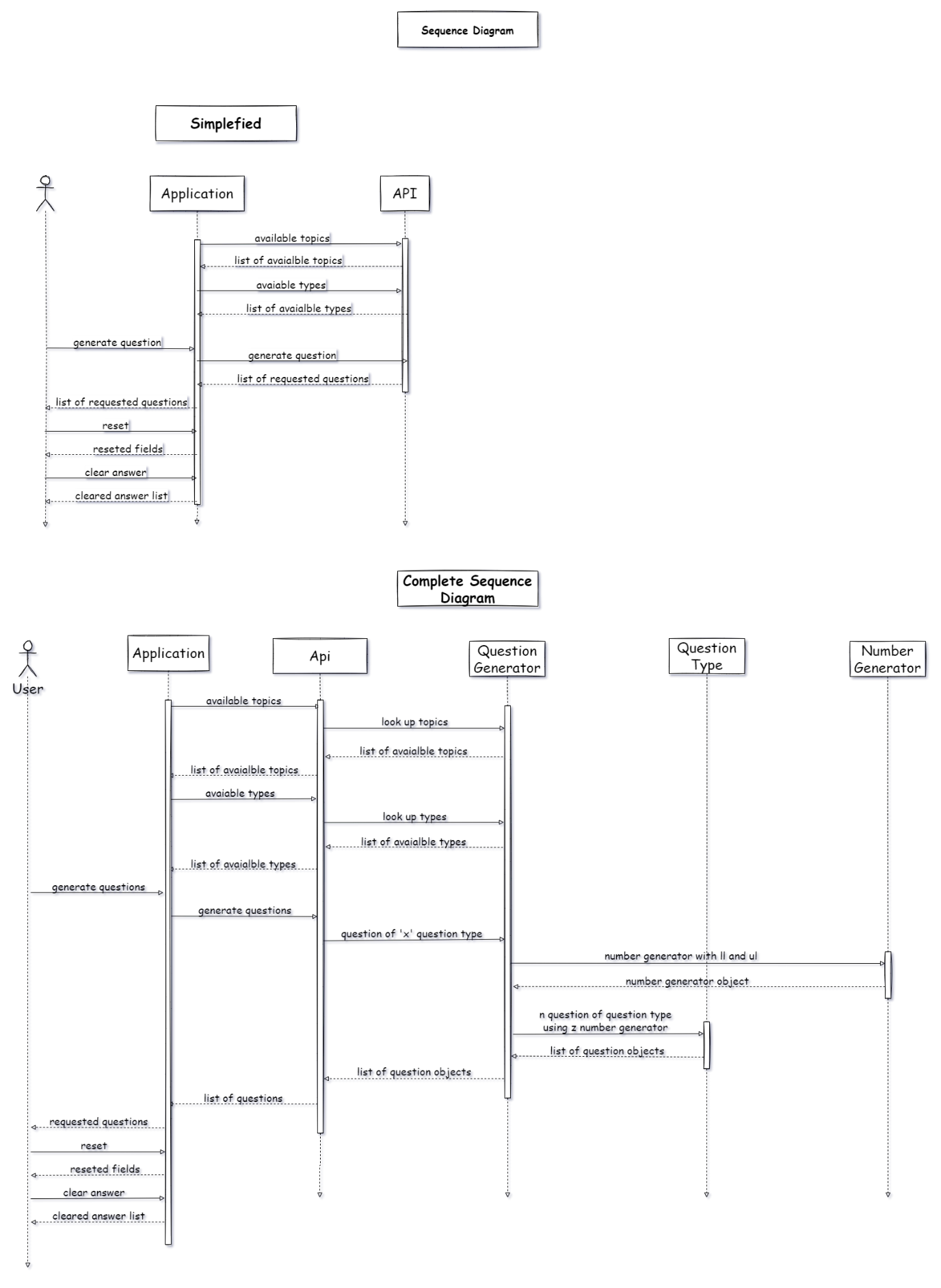
****

**Sequence Diagram**

Sequence Diagram is a diagram that explains the order of interactions that the any actor is going to have with the system. The diagram shows how—and in what order—a group of objects works together.

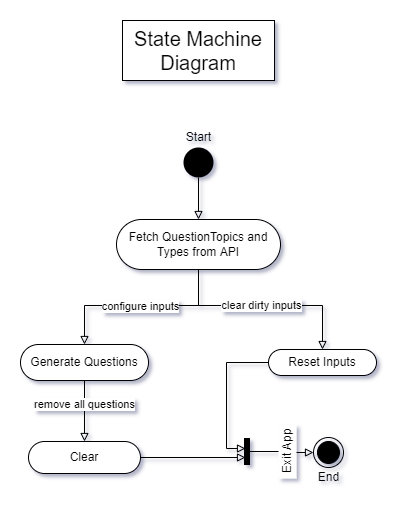


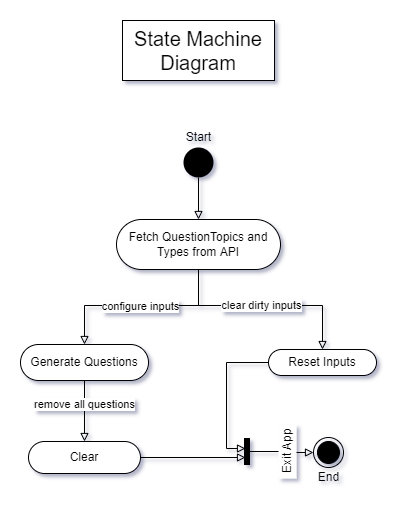
****

****

**State Machine Diagram**

A state machine is any device that stores the status of an object at a given time and can change status or cause other actions based on the input it receives. States refer to the different combinations of information that an object can hold, not how the object behaves.

****

****

**System Coding**

**main.py**

# math\_gen\_api/main.py

# In-built imports

from typing import Any, Dict, Optional, Tuple

# Third-party imports

from flask import Flask, jsonify, request, render\_template

from flask.wrappers import Response, Request

# Sys-Paths for Relative Imports

import sys

from os.path import dirname, abspath

package\_path = dirname(dirname(abspath(\_\_file\_\_)))

if(package\_path not in sys.path): sys.path.insert(0, package\_path)

# Relative imports

from math\_gen\_api.question\_generator import COMBINE\_LOOKUP, Question\_Generator

from question\_strategies import question

##############################

# Flask API Init

##############################

app = Flask(\_\_name\_\_)

app.config["ENV"] = "development"

# global question generator factory

def createErrorResponse(err\_message, err\_code):

    return Response(err\_message, err\_code)

##############################

# Documentation Routes

##############################

@app.route("/")

def index(): # url navigation info

    return render\_template("index.html")

@app.route("/docs")

def docs(): # url navigation info

    return render\_template("question\_docs.html")

@app.route("/available topics", methods=["GET", "OPTIONS"])

def topicOptions():

    options = COMBINE\_LOOKUP.keys()

    if options == None: return createErrorResponse("Invalid Topic", 404)

    return jsonify(list(options))

@app.route("/available types", methods=["GET", "OPTIONS"])

def typeOptions():

    q\_topic = request.args.get("topic")

    if q\_topic == None: return createErrorResponse("Invalid Argument", 400)

    options = COMBINE\_LOOKUP.get(q\_topic)

    if options == None: return createErrorResponse("Invalid Topic", 404)

    return jsonify(list(options.keys()))

##############################

# Question Routes

##############################

@app.route("/question")

def questionRoute():

    request\_data = request.get\_json()

    # if request\_data == None: return createErrorResponse("Bad Request", 400)

    if request\_data == None: request\_data = request.args.to\_dict()

    if request\_data is None : return render\_template("question\_docs.html")

    print("request\_data", request\_data)

    q\_topic:Optional[str] = request\_data.get("q\_topic", None)

    q\_type:Optional[str] = request\_data.get("q\_type", None)

    noq:int = int(request\_data.get("noq", 1))

    ll:Optional[int] = request\_data.get("ll", None)

    if ll != None: ll= int(ll)

    ul:Optional[int] = request\_data.get("ul", None)

    if ul != None: ul= int(ul)

    if not all((q\_topic,q\_type)): return createErrorResponse("Error - Bad Request", 400)

    if ll> ul: return createErrorResponse("Error - Bad Limit", 400)

    question\_list:list = []

    for \_ in range(noq):

        question\_generator:Optional[question.QuestionType] = Question\_Generator(q\_topic, q\_type, ll, ul)

        if question\_generator == None: return createErrorResponse("Error - Bad Arguments 1", 400)

        try:

            q:question.Question = question\_generator.generate\_question()

        except Exception as err:

            print(err.args)

            return createErrorResponse("Error - Bad Result 2", 400)

        question\_list.append(q.toDict())

    return jsonify(question\_list)

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

    app.run(debug=False)

**question\_generator.py**

# math\_gen\_api/question\_generator.py

# In-built imports

from typing import Any, Optional, Type

# Third-party imports

# Sys-Paths for Relative Imports

import sys

from os.path import dirname, abspath

package\_path = dirname(abspath(\_\_file\_\_))

if(package\_path not in sys.path): sys.path.insert(0, package\_path)

# Relative imports

from number\_gen import integer\_number

from question\_strategies import addition, subtraction, multiplication, division

from question\_strategies import lcm, hcf, quadratic, linear2var, factors, square, factorial

from question\_strategies import permutation, combination , fibonacci, profit, loss

COMBINE\_LOOKUP:dict[str, dict[str, Type[Any]]] = {

    "addition": addition.TYPE\_LOOKUP,

    "subtraction": subtraction.TYPE\_LOOKUP,

    "multiplication": multiplication.TYPE\_LOOKUP,

    "division": division.TYPE\_LOOKUP,

    "lcm": lcm.TYPE\_LOOKUP,

    "hcf": hcf.TYPE\_LOOKUP,

    "quadratic": quadratic.TYPE\_LOOKUP,

    "linear2var": linear2var.TYPE\_LOOKUP,

    "factors": factors.TYPE\_LOOKUP,

    "square": square.TYPE\_LOOKUP,

    "factorial": factorial.TYPE\_LOOKUP,

    "permutation": permutation.TYPE\_LOOKUP,

    "combination": combination.TYPE\_LOOKUP,

    "fibonacci": fibonacci.TYPE\_LOOKUP,

    "profit": profit.TYPE\_LOOKUP,

    "loss": loss.TYPE\_LOOKUP

}

def Question\_Generator(q\_topic:str, q\_type:str, ll:Optional[int], ul:Optional[int]):

    question\_generator\_cls = COMBINE\_LOOKUP.get(q\_topic, {}).get(q\_type, None)

    if question\_generator\_cls == None: return None

    question\_generator = question\_generator\_cls(integer\_number.RangedIntegerNumberGenerator(ll, ul))

    return question\_generator

**number\_generator.py**

# In-built imports

import sys

from os.path import dirname, abspath

from abc import ABC, abstractmethod

from typing import Optional, TypeVar

# Third-party imports

# Sys-Paths for Relative Imports

import sys

from os.path import dirname, abspath

\_package\_path = dirname(dirname(abspath(\_\_file\_\_)))

if(\_package\_path not in sys.path): sys.path.insert(0, \_package\_path)

# Relative imports

numberType = TypeVar("numberType", int, float)

class NumberGenerator(ABC):

    """Generic Abstract Class for Number Generator

    Args:

        ABC (\_type\_): \_description\_

    """

    @abstractmethod

    def number(self, is\_negative:Optional[bool]=False, is\_zero:Optional[bool]=False):

        ...

class RangedNumberGenerator(NumberGenerator, ABC):

    """Abstract class for Ranged Numbercler Generatoration

    \_\_init\_\_ takes 2 arguments lower\_limit and upper\_limit

    lower\_limit and upper\_limit are of type int or float

    Args:

        ABC (\_type\_): abstract base class

    """

    def \_\_init\_\_(self, lower\_limit:numberType, upper\_limit:numberType) -> None:

        if lower\_limit > upper\_limit: raise Exception("Lower Limit should be smaller than Upper Limit")

        self.\_\_lower\_limit = lower\_limit

        self.\_\_upper\_limit = upper\_limit

    @property

    def lower\_limit(self):

        return self.\_\_lower\_limit

    @lower\_limit.setter

    def lower\_limit(self, ll):

        if ll > self.upper\_limit: raise Exception("Lower Limit should be smaller than Upper Limit")

        self.\_\_lower\_limit = ll

    @property

    def upper\_limit(self):

        return self.\_\_upper\_limit

    @upper\_limit.setter

    def upper\_limit(self, ul):

        if ul < self.lower\_limit: raise Exception("Upper Limit should be larger than Lower Limit")

        self.\_\_upper\_limit = ul

    @abstractmethod

    def number(self):

        ...

**Integer\_number.py**

# In-built imports

import random

from typing import Callable, Optional

# Third-party imports

# Sys-Paths for Relative Imports

import sys

from os.path import dirname, abspath

package\_path = dirname(dirname(abspath(\_\_file\_\_)))

if(package\_path not in sys.path): sys.path.insert(0, package\_path)

# Relative imports

from number\_gen import number\_generator

def intRanged(num\_func:Callable):

    """Decorator for number method in IntegerNumberGenerators

    checks for 'is\_negative' and 'is\_zero' keyword arguments

    if is\_negative is True it will return negative of generated number

    if is\_zero is True it will return 0

    Args:

        num\_func (Callable): \_description\_

    """

    def wrapper(\*args, is\_negative:Optional[bool]=False, is\_zero:Optional[bool]=False, ):

        if is\_zero: return 0

        num = num\_func(\*args)

        if is\_negative: num \*= -1

        return num

    return wrapper

class RangedIntegerNumberGenerator(number\_generator.RangedNumberGenerator):

    """Ranged Ineger Number Generator

    Generates numbers between a specified lower limit and upper limit.

    lower and upper limits are passed as arguments to the \_\_init\_\_ method.

    Args:

        number\_generator (\_type\_): module that holds base classes for number generators

    """

    def \_\_init\_\_(self, lower\_limit:Optional[int]=None, upper\_limit:Optional[int]=None) -> None:

        if not lower\_limit: lower\_limit = 1

        if not upper\_limit: upper\_limit = 1000

        super().\_\_init\_\_(lower\_limit, upper\_limit)

    @intRanged

    def number(self):

        num = random.randint(int(self.lower\_limit), int(self.upper\_limit))

        return num

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

    def main():

        # Code Here

        ...

    main()

**question.py**

# question\_strategies/question.py

# In-built imports

import sys

from os.path import dirname, abspath

from abc import ABC, abstractmethod

from typing import Union

# import json, xml

# Third-party imports

# Sys-Paths for Relative Imports

import sys

from os.path import dirname, abspath

package\_path = dirname(dirname(abspath(\_\_file\_\_)))

if(package\_path not in sys.path): sys.path.insert(0, package\_path)

# Relative imports

from number\_gen import integer\_number, floating\_number

numGenType = Union[integer\_number.RangedIntegerNumberGenerator, floating\_number.RangedFloatingNumberGenerator]

class Question:

    """This is a Generic Question class

    """

    def \_\_init\_\_(self, qstring: str, answer, question\_type: str) -> None:

        self.qstring = qstring

        self.answer = answer

        self.type = question\_type

    def \_\_repr\_\_(self) -> str:

        return f"[Question] {self.qstring} \tAnswer: {self.answer}"

    def toDict(self):

        """Returns Dictionary Object of Question

        Returns:

            dict: keys [ question\_string, answer, type ]

        """

        q = {

            "question\_string": self.qstring,

            "answer": self.answer,

            "type": self.type

        }

        return q

    def toJson(self):

        """Returns JSON Object of Question

        Returns:

            dict: keys [ question\_string, answer, type ]

        """

        q = {

            "question\_string": self.qstring,

            "answer": self.answer,

            "type": self.type

        }

        # convert to JSON

        return q

    def toXml(self):

        """Returns XML Object of Question

        Returns:

            dict: keys [ question\_string, answer, type ]

        """

        q = {

            "question\_string": self.qstring,

            "answer": self.answer,

            "type": self.type

        }

        # convert to XML

        return q

class QuestionType(ABC):

    """Abstract class for all QuestionTypes

    If you want to make a new question it must extend this class

    Args:

        ABC (\_type\_): abstract base class

    """

    @abstractmethod

    def generate\_question(self) -> Question:

        ...

**addition.py**

# question\_strategies/addition.py

# In-built imports

from typing import Type

# Third-party imports

# Sys-Paths for Relative Imports

import sys

from os.path import dirname, abspath

package\_path = dirname(dirname(abspath(\_\_file\_\_)))

if(package\_path not in sys.path): sys.path.insert(0, package\_path)

# Relative imports

from question\_strategies import question

class AdditionQuestionType1(question.QuestionType):

    Q\_TYPE = "AdditionType1"

    INIT\_VARIABLES= {

        "number\_of\_nums": "int"

    }

    def \_\_init\_\_(self, number\_generator\_cls:question.numGenType, number\_of\_nums:int=2) -> None:

        super().\_\_init\_\_()

        self.number\_generator\_obj = number\_generator\_cls

        self.operator = "+"

        if number\_of\_nums < 2: raise Exception("number\_of\_nums must be 2 or greater")

        self.number\_of\_nums = number\_of\_nums

    def generate\_question(self) -> question.Question:

        format\_string = f" {self.operator} ".join(["{}" for \_ in range(self.number\_of\_nums)])

        num\_list = [self.number\_generator\_obj.number() for \_ in range(self.number\_of\_nums)]

        question\_string = format\_string.format(\*num\_list)

        return question.Question(question\_string, eval(question\_string), self.Q\_TYPE)

TYPE\_LOOKUP:dict[str, Type[question.QuestionType]] = {

    "normal": AdditionQuestionType1

}

**combination.py**

# question\_strategies/addition.py

# In-built imports

from typing import Type

import math

# Third-party imports

# Sys-Paths for Relative Imports

import sys

from os.path import dirname, abspath

package\_path = dirname(dirname(abspath(\_\_file\_\_)))

if(package\_path not in sys.path): sys.path.insert(0, package\_path)

# Relative imports

from question\_strategies import question

class CombinationQuestionType1(question.QuestionType):

    Q\_TYPE = "CombinationType1"

    INIT\_VARIABLES= {

        "number\_of\_nums": "int"

    }

    def \_\_init\_\_(self, number\_generator\_cls:question.numGenType, number\_of\_nums:int=2) -> None:

        super().\_\_init\_\_()

        self.number\_generator\_obj = number\_generator\_cls

        self.operator = "+"

        if number\_of\_nums < 2: raise Exception("number\_of\_nums must be 2 or greater")

        self.number\_of\_nums = number\_of\_nums

    def generate\_question(self) -> question.Question:

        n ,r = sorted([self.number\_generator\_obj.number() for i in range(2)], reverse=True)

        c = math.factorial(n) / math.factorial(r) \* math.factorial(n - r)

        question\_string = f"Find the number of combinations when n={n} and r={r}"

        return question.Question(question\_string, c, self.Q\_TYPE)

TYPE\_LOOKUP:dict[str, Type[question.QuestionType]] = {

    "normal": CombinationQuestionType1

}

**division.py**

# question\_strategies/division.py

# In-built imports

from typing import Type, Union

# Third-party imports

# Sys-Paths for Relative Imports

import sys

from os.path import dirname, abspath

package\_path = dirname(dirname(abspath(\_\_file\_\_)))

if(package\_path not in sys.path): sys.path.insert(0, package\_path)

# Relative imports

from question\_strategies import question

class DivisionQuestionType1(question.QuestionType):

    Q\_TYPE = "Division\_Type1"

    def \_\_init\_\_(self, number\_generator\_cls:question.numGenType) -> None:

        super().\_\_init\_\_()

        self.number\_generator\_obj = number\_generator\_cls

        self.operator = "/"

        self.number\_of\_nums = 2

    def generate\_question(self):

        format\_string = f" {self.operator} ".join(["{}" for \_ in range(self.number\_of\_nums)])

        multiple\_number\_func = lambda x: [x \* self.number\_generator\_obj.number(), x]

        num\_list = multiple\_number\_func(self.number\_generator\_obj.number())

        question\_string = format\_string.format(\*num\_list)

        return question.Question(question\_string, eval(question\_string), self.Q\_TYPE.title())

class DivisionQuestionType2(question.QuestionType):

    Q\_TYPE = "Division\_Type2"

    INIT\_VARIABLES= {}

    def \_\_init\_\_(self, number\_generator\_cls:question.numGenType) -> None:

        super().\_\_init\_\_()

        self.number\_generator\_obj = number\_generator\_cls

        self.operator = "/"

        self.number\_of\_nums = 2

    def generate\_question(self):

        format\_string = f" {self.operator} ".join(["{}" for \_ in range(self.number\_of\_nums)])

        multiple\_number\_func = lambda x: [x \* self.number\_generator\_obj.number(is\_negative=True), x]

        num\_list = multiple\_number\_func(self.number\_generator\_obj.number())

        question\_string = format\_string.format(\*num\_list)

        return question.Question(question\_string, eval(question\_string), self.Q\_TYPE.title())

class DivisionQuestionType3(question.QuestionType):

    Q\_TYPE = "Division\_Type2"

    def \_\_init\_\_(self, number\_generator\_cls:question.numGenType) -> None:

        super().\_\_init\_\_()

        self.number\_generator\_obj = number\_generator\_cls

        self.operator = "/"

        self.number\_of\_nums = 2

    def generate\_question(self):

        format\_string = f" {self.operator} ".join(["{}" for \_ in range(self.number\_of\_nums)])

        multiple\_number\_func = lambda x: [x \* self.number\_generator\_obj.number(), x]

        num\_list = multiple\_number\_func(self.number\_generator\_obj.number(is\_negative=True))

        question\_string = format\_string.format(\*num\_list)

        return question.Question(question\_string, eval(question\_string), self.Q\_TYPE.title())

TYPE\_LOOKUP:dict[str, Type[question.QuestionType]] = {

    "positive": DivisionQuestionType1,

    "negative": DivisionQuestionType2,

    "double negative": DivisionQuestionType3

}

**factorial.py**

# question\_strategies/addition.py

# In-built imports

from typing import Type

import math

# Third-party imports

# Sys-Paths for Relative Imports

import sys

from os.path import dirname, abspath

package\_path = dirname(dirname(abspath(\_\_file\_\_)))

if(package\_path not in sys.path): sys.path.insert(0, package\_path)

# Relative imports

from question\_strategies import question

class FactorialQuestionType1(question.QuestionType):

    Q\_TYPE = "FactorialType1"

    INIT\_VARIABLES= {

        "number\_of\_nums": "int"

    }

    def \_\_init\_\_(self, number\_generator\_cls:question.numGenType, number\_of\_nums:int=1) -> None:

        super().\_\_init\_\_()

        self.number\_generator\_obj = number\_generator\_cls

        self.operator = "+"

        if number\_of\_nums < 1: raise Exception("number\_of\_nums must be 2 or greater")

        self.number\_of\_nums = number\_of\_nums

    def generate\_question(self) -> question.Question:

        format\_string = f"Find Factorial of: " + f"{self.operator}".join(["{}" for \_ in range(self.number\_of\_nums)])

        num\_list = [self.number\_generator\_obj.number() for \_ in range(self.number\_of\_nums)]

        question\_string = format\_string.format(\*num\_list)

        return question.Question(question\_string, self.fact(num\_list), self.Q\_TYPE)

    def fact(self, num\_list):

        return [math.factorial(i) for i in num\_list]

TYPE\_LOOKUP:dict[str, Type[question.QuestionType]] = {

    "normal": FactorialQuestionType1

}

**factors.py**

# question\_strategies/factors.py

# In-built imports

from typing import Type

import math, random

# Third-party imports

# Sys-Paths for Relative Imports

import sys

from os.path import dirname, abspath

package\_path = dirname(dirname(abspath(\_\_file\_\_)))

if(package\_path not in sys.path): sys.path.insert(0, package\_path)

# Relative imports

from question\_strategies import question

class FactorsQuestionType1(question.QuestionType):

    Q\_TYPE = "Factors\_Type1"

    INIT\_VARIABLES= {

        "number\_of\_nums": "int"

    }

    def \_\_init\_\_(self, number\_generator\_cls:question.numGenType, number\_of\_nums:int=2) -> None:

        super().\_\_init\_\_()

        self.number\_generator\_obj = number\_generator\_cls

        self.operator = "\*"

        if number\_of\_nums < 2: raise Exception("number\_of\_nums must be 2 or greater")

        self.number\_of\_nums = number\_of\_nums

    def generate\_question(self) -> question.Question:

        num = self.number\_generator\_obj.number()

        factors = self.factorlist(num)

        while len(factors) <= 2:

            num = self.number\_generator\_obj.number()

            factors = self.factorlist(num)

        num\_set1 = random.choice(factors)

        factors.remove(num\_set1)

        num\_set2 = random.choice(factors)

        question\_string = f"Find the Missing Factor: {num\_set1[0]}x{'?'}  = {num\_set2[0]}x{num\_set2[1]}"

        return question.Question(question\_string, num\_set1[1], self.Q\_TYPE.title())

    def factorlist(self, num:int):

        fac\_list = []

        for i in range(1, int(math.sqrt(num)+1)):

            if num % i == 0:

                quotient = num // i

                if ((quotient, i) in fac\_list): continue

                fac\_list.append((i, quotient))

        return list(fac\_list)

class FactorsQuestionType2(question.QuestionType):

    Q\_TYPE = "Factors\_Type2"

    INIT\_VARIABLES= {

        "number\_of\_nums": "int"

    }

    def \_\_init\_\_(self, number\_generator\_cls:question.numGenType, number\_of\_nums:int=2) -> None:

        super().\_\_init\_\_()

        self.number\_generator\_obj = number\_generator\_cls

        self.operator = "\*"

        if number\_of\_nums < 2: raise Exception("number\_of\_nums must be 2 or greater")

        self.number\_of\_nums = number\_of\_nums

    def generate\_question(self) -> question.Question:

        num = self.number\_generator\_obj.number()

        question\_string = f"List all Factors of : {num}"

        return question.Question(question\_string, self.factorlist(num), self.Q\_TYPE.title())

    def factorlist(self, num:int):

        fac\_list = []

        for i in range(1, int(num)):

            if num % i == 0:

                fac\_list.append((i))

        return list(fac\_list)

TYPE\_LOOKUP:dict[str, Type[question.QuestionType]] = {

    "missing": FactorsQuestionType1,

    "list": FactorsQuestionType2

}

**fibonacci.py**

# question\_strategies/addition.py

# In-built imports

from typing import Type

# Third-party imports

# Sys-Paths for Relative Imports

import sys

from os.path import dirname, abspath

package\_path = dirname(dirname(abspath(\_\_file\_\_)))

if(package\_path not in sys.path): sys.path.insert(0, package\_path)

# Relative imports

from question\_strategies import question

class FibonacciQuestionType1(question.QuestionType):

    Q\_TYPE = "FibonacciType1"

    INIT\_VARIABLES= {

        "number\_of\_nums": "int"

    }

    def \_\_init\_\_(self, number\_generator\_cls:question.numGenType, number\_of\_nums:int=1) -> None:

        super().\_\_init\_\_()

        self.number\_generator\_obj = number\_generator\_cls

        self.operator = ","

        if number\_of\_nums < 1: raise Exception("number\_of\_nums must be 2 or greater")

        self.number\_of\_nums = number\_of\_nums

    def generate\_question(self) -> question.Question:

        format\_string = f"Find nth Fibonacci Term: " + f"{self.operator}".join(["{}" for \_ in range(self.number\_of\_nums)])

        num\_list = [self.number\_generator\_obj.number() for \_ in range(self.number\_of\_nums)]

        question\_string = format\_string.format(\*num\_list)

        return question.Question(question\_string, self.fibo(num\_list), self.Q\_TYPE)

    def fibo(self, num\_list):

        new\_list = []

        for n in num\_list:

            a = 0

            b = 1

            if n < 0:

                print("Incorrect input")

            elif n == 0:

                new\_list.append(a)

                continue

            elif n == 1:

                new\_list.append(b)

                continue

            else:

                for \_ in range(2,n+1):

                    c = a + b

                    a = b

                    b = c

                new\_list.append(b)

                continue

        return new\_list

TYPE\_LOOKUP:dict[str, Type[question.QuestionType]] = {

    "nth term": FibonacciQuestionType1

}

**hcf.py**

# question\_strategies/hcf.py

# In-built imports

from typing import Type

# Third-party imports

# Sys-Paths for Relative Imports

import sys

from os.path import dirname, abspath

package\_path = dirname(dirname(abspath(\_\_file\_\_)))

if(package\_path not in sys.path): sys.path.insert(0, package\_path)

# Relative imports

from question\_strategies import question

class HCFQuestionType1(question.QuestionType):

    Q\_TYPE = "HCF\_Type1"

    INIT\_VARIABLES= {

        "number\_of\_nums": "int"

    }

    def \_\_init\_\_(self, number\_generator\_cls:question.numGenType, number\_of\_nums:int=2) -> None:

        super().\_\_init\_\_()

        self.number\_generator\_obj = number\_generator\_cls

        self.operator = ","

        if number\_of\_nums < 2: raise Exception("number\_of\_nums must be 2 or greater")

        self.number\_of\_nums = number\_of\_nums

    def generate\_question(self) -> question.Question:

        format\_string = f" {self.operator} ".join(["{}" for \_ in range(self.number\_of\_nums)])

        num\_list = [self.number\_generator\_obj.number() for \_ in range(self.number\_of\_nums)]

        question\_string = "Find HCF of: " + format\_string.format(\*num\_list)

        return question.Question(question\_string, self.findHCF(num\_list), self.Q\_TYPE)

    def findHCF(self, num\_list:list):

        def gcd(n,d):

            if (d == 0):

                return n

            return gcd(d, n%d)

        def gcdArray(num\_list:list):

            num\_list.sort()

            hcf = num\_list.pop()

            while len(num\_list):

                # print(hcf, num\_list[0])

                hcf = gcd(hcf, num\_list[0])

                num\_list.pop(0)

            return hcf

        return gcdArray(num\_list)

TYPE\_LOOKUP:dict[str, Type[question.QuestionType]] = {

    "normal": HCFQuestionType1

}

**lcm.py**

# question\_strategies/lcm.py

# In-built imports

from typing import Type, Union

# Third-party imports

# Sys-Paths for Relative Imports

import sys

from os.path import dirname, abspath

package\_path = dirname(dirname(abspath(\_\_file\_\_)))

if(package\_path not in sys.path): sys.path.insert(0, package\_path)

# Relative imports

from question\_strategies import question

class LCMQuestionType1(question.QuestionType):

    Q\_TYPE = "LCM\_Type1"

    INIT\_VARIABLES= {

        "number\_of\_nums": "int"

    }

    def \_\_init\_\_(self, number\_generator\_cls:question.numGenType, number\_of\_nums:int=2) -> None:

        super().\_\_init\_\_()

        self.number\_generator\_obj = number\_generator\_cls

        self.operator = ","

        if number\_of\_nums < 2: raise Exception("number\_of\_nums must be 2 or greater")

        self.number\_of\_nums = number\_of\_nums

    def generate\_question(self) -> question.Question:

        format\_string = f" {self.operator} ".join(["{}" for \_ in range(self.number\_of\_nums)])

        num\_list = [self.number\_generator\_obj.number() for \_ in range(self.number\_of\_nums)]

        question\_string = "Find LCM of: " + format\_string.format(\*num\_list)

        return question.Question(question\_string, self.findLCM(num\_list), self.Q\_TYPE)

    def gcd(self, a:int, b:int) -> int:

        if (b == 0):

            return a

        return self.gcd(b, a % b)

    def findLCM(self, arr:list[int]) -> Union[int, float]:

        ans = arr[0]

        for i in range(1, len(arr)):

            ans = (arr[i] \* ans) / self.gcd(arr[i], ans)

        return ans

TYPE\_LOOKUP:dict[str, Type[question.QuestionType]] = {

    "normal": LCMQuestionType1

}

**linear2var.py**

# question\_strategies/linear2var.py

# In-built imports

from typing import Type

# Third-party imports

# Sys-Paths for Relative Imports

import sys

from os.path import dirname, abspath

package\_path = dirname(dirname(abspath(\_\_file\_\_)))

if(package\_path not in sys.path): sys.path.insert(0, package\_path)

# Relative imports

from question\_strategies import question

class Linear2VarQuestionType1(question.QuestionType):

    Q\_TYPE = "Linear\_2\_Var\_Type1"

    INIT\_VARIABLES= {}

    def \_\_init\_\_(self, number\_generator\_cls:question.numGenType) -> None:

        super().\_\_init\_\_()

        self.number\_generator\_obj = number\_generator\_cls

    def generate\_question(self):

        format\_string = "{}x + {}y + {} = 0;{}x + {}y + {} = 0;"

        p1 = [self.number\_generator\_obj.number() for \_ in range(2)]

        p2 = [self.number\_generator\_obj.number() for \_ in range(2)]

        sol = [ self.number\_generator\_obj.number() for \_ in range(2)]

        c1 = self.generate\_coefficient(p1, sol)

        c2 = self.generate\_coefficient(p2, sol)

        question\_string = format\_string.format(\*c1, \*c2)

        return question.Question(question\_string, sol, self.Q\_TYPE.title())

    def generate\_coefficient(self, point, ans\_point):

        a = (ans\_point[1] - point[1])

        b = (ans\_point[0] - point[0]) \* -1

        c = -(point[0] \* (ans\_point[1] - point[1])) + (point[1] \* (ans\_point[0] - point[0]))

        if(a < 0):

            a\*= -1

            b\*= -1

            c\*= -1

        return [a, b, c]

TYPE\_LOOKUP:dict[str, Type[question.QuestionType]] = {

    "normal": Linear2VarQuestionType1

}

**loss.py**

# question\_strategies/addition.py

# In-built imports

from typing import Type

import math

# Third-party imports

# Sys-Paths for Relative Imports

import sys

from os.path import dirname, abspath

package\_path = dirname(dirname(abspath(\_\_file\_\_)))

if(package\_path not in sys.path): sys.path.insert(0, package\_path)

# Relative imports

from question\_strategies import question

class LossQuestionType1(question.QuestionType):

    Q\_TYPE = "LossType1"

    INIT\_VARIABLES= {

        "number\_of\_nums": "int"

    }

    def \_\_init\_\_(self, number\_generator\_cls:question.numGenType, number\_of\_nums:int=2) -> None:

        super().\_\_init\_\_()

        self.number\_generator\_obj = number\_generator\_cls

        if number\_of\_nums < 2: raise Exception("number\_of\_nums must be 2 or greater")

        self.number\_of\_nums = number\_of\_nums

    def generate\_question(self) -> question.Question:

        format\_string = "Find Loss Percentage When Cost Price is {} and Sell Price is {}"

        num\_list = [self.number\_generator\_obj.number() for \_ in range(2)]

        num\_list.sort(reverse=True)

        cp, sp = num\_list

        percentage = abs(self.percent(cp, sp))

        question\_string = format\_string.format(cp, sp)

        return question.Question(question\_string, percentage, self.Q\_TYPE)

    def percent(self, cp, sp):

        return round(((sp - cp) / sp) \* 100, 2)

class LossQuestionType2(question.QuestionType):

    Q\_TYPE = "LossType2"

    INIT\_VARIABLES= {

        "number\_of\_nums": "int"

    }

    def \_\_init\_\_(self, number\_generator\_cls:question.numGenType, number\_of\_nums:int=2) -> None:

        super().\_\_init\_\_()

        self.number\_generator\_obj = number\_generator\_cls

        if number\_of\_nums < 2: raise Exception("number\_of\_nums must be 2 or greater")

        self.number\_of\_nums = number\_of\_nums

    def generate\_question(self) -> question.Question:

        format\_string = "Find Cost Price When Loss Percentage is {} and Sell Price is {}"

        num\_list = [self.number\_generator\_obj.number() for \_ in range(2)]

        num\_list.sort(reverse=True)

        cp, sp = num\_list

        percentage = abs(self.percent(cp, sp))

        question\_string = format\_string.format(percentage, sp)

        return question.Question(question\_string, cp, self.Q\_TYPE)

    def percent(self, cp, sp):

        return round(((sp - cp) / sp) \* 100, 2)

class LossQuestionType3(question.QuestionType):

    Q\_TYPE = "LossType3"

    INIT\_VARIABLES= {

        "number\_of\_nums": "int"

    }

    def \_\_init\_\_(self, number\_generator\_cls:question.numGenType, number\_of\_nums:int=2) -> None:

        super().\_\_init\_\_()

        self.number\_generator\_obj = number\_generator\_cls

        if number\_of\_nums < 2: raise Exception("number\_of\_nums must be 2 or greater")

        self.number\_of\_nums = number\_of\_nums

    def generate\_question(self) -> question.Question:

        format\_string = "Find Sell Price When Loss Percentage is {} and Cost Price is {}"

        num\_list = [self.number\_generator\_obj.number() for \_ in range(2)]

        num\_list.sort(reverse=True)

        cp, sp = num\_list

        percentage = abs(self.percent(cp, sp))

        question\_string = format\_string.format(percentage, cp)

        return question.Question(question\_string, sp, self.Q\_TYPE)

    def percent(self, cp, sp):

        return round(((sp - cp) / sp) \* 100, 2)

TYPE\_LOOKUP:dict[str, Type[question.QuestionType]] = {

    "Find Profit": LossQuestionType1,

    "FInd Cost": LossQuestionType2,

    "Find Sell": LossQuestionType3

}

**multiplication.py**

# question\_strategies/multiplication.py

# In-built imports

import random

from typing import Type, Union

# Third-party imports

# Sys-Paths for Relative Imports

import sys

from os.path import dirname, abspath

package\_path = dirname(dirname(abspath(\_\_file\_\_)))

if(package\_path not in sys.path): sys.path.insert(0, package\_path)

# Relative imports

from question\_strategies import question

class MultiplicationQuestionType1(question.QuestionType):

    Q\_TYPE = "Multiplication\_Type1"

    INIT\_VARIABLES= {

        "number\_of\_nums": "int"

    }

    def \_\_init\_\_(self, number\_generator\_cls:question.numGenType, number\_of\_nums:int=2) -> None:

        super().\_\_init\_\_()

        self.number\_generator\_obj = number\_generator\_cls

        self.operator = "\*"

        if number\_of\_nums < 2: raise Exception("number\_of\_nums must be 2 or greater")

        self.number\_of\_nums = number\_of\_nums

    def generate\_question(self):

        format\_string = f" {self.operator} ".join(["{}" for \_ in range(self.number\_of\_nums)])

        num\_list = [self.number\_generator\_obj.number() for \_ in range(self.number\_of\_nums)]

        question\_string = format\_string.format(\*num\_list)

        return question.Question(question\_string, eval(question\_string), self.Q\_TYPE.title())

class MultiplicationQuestionType2(question.QuestionType):

    Q\_TYPE = "Multiplication\_Type2"

    INIT\_VARIABLES= {

        "number\_of\_nums": "int"

    }

    def \_\_init\_\_(self, number\_generator\_cls:question.numGenType, number\_of\_nums:int=2) -> None:

        super().\_\_init\_\_()

        self.number\_generator\_obj = number\_generator\_cls

        self.operator = "\*"

        if number\_of\_nums < 2: raise Exception("number\_of\_nums must be 2 or greater")

        self.number\_of\_nums = number\_of\_nums

    def generate\_question(self):

        format\_string = f" {self.operator} ".join(["{}" for \_ in range(self.number\_of\_nums)])

        num\_list = [self.number\_generator\_obj.number() for \_ in range(self.number\_of\_nums//2)]

        num\_list += [self.number\_generator\_obj.number(is\_negative=True) for \_ in range(self.number\_of\_nums - self.number\_of\_nums//2)]

        random.shuffle(num\_list)

        question\_string = format\_string.format(\*num\_list)

        return question.Question(question\_string, eval(question\_string), self.Q\_TYPE.title())

TYPE\_LOOKUP:dict[str, Type[question.QuestionType]] = {

    "positive": MultiplicationQuestionType1,

    "negative": MultiplicationQuestionType2

}

**permutation.py**

# question\_strategies/addition.py

# In-built imports

from typing import Type

import math

# Third-party imports

# Sys-Paths for Relative Imports

import sys

from os.path import dirname, abspath

package\_path = dirname(dirname(abspath(\_\_file\_\_)))

if(package\_path not in sys.path): sys.path.insert(0, package\_path)

# Relative imports

from question\_strategies import question

class PermuataionQuestionType1(question.QuestionType):

    Q\_TYPE = "PermuataionType1"

    INIT\_VARIABLES= {

        "number\_of\_nums": "int"

    }

    def \_\_init\_\_(self, number\_generator\_cls:question.numGenType) -> None:

        super().\_\_init\_\_()

        self.number\_generator\_obj = number\_generator\_cls

    def generate\_question(self) -> question.Question:

        n ,r = sorted([self.number\_generator\_obj.number() for i in range(2)], reverse=True)

        p = math.factorial(n) / math.factorial(n - r)

        print(p)

        question\_string = f"Find the number of permutaions when n={n} and r={r} repeatation not allowed"

        return question.Question(question\_string, p, self.Q\_TYPE)

class PermuataionQuestionType2(question.QuestionType):

    Q\_TYPE = "PermuataionType2"

    INIT\_VARIABLES= {

        "number\_of\_nums": "int"

    }

    def \_\_init\_\_(self, number\_generator\_cls:question.numGenType) -> None:

        super().\_\_init\_\_()

        self.number\_generator\_obj = number\_generator\_cls

    def generate\_question(self) -> question.Question:

        n ,r = sorted([self.number\_generator\_obj.number() for i in range(2)], reverse=True)

        p = n\*\*r

        question\_string = f"Find the number of permutaions when n={n} and r={r} repeatation allowed"

        return question.Question(question\_string, p, self.Q\_TYPE)

TYPE\_LOOKUP:dict[str, Type[question.QuestionType]] = {

    "without repeatation": PermuataionQuestionType1, # without repeatation

    "with repeatation": PermuataionQuestionType2 # with repeatation

}

**profit.py**

# question\_strategies/addition.py

# In-built imports

from typing import Type

import math

# Third-party imports

# Sys-Paths for Relative Imports

import sys

from os.path import dirname, abspath

package\_path = dirname(dirname(abspath(\_\_file\_\_)))

if(package\_path not in sys.path): sys.path.insert(0, package\_path)

# Relative imports

from question\_strategies import question

class ProfitQuestionType1(question.QuestionType):

    Q\_TYPE = "ProfitType1"

    INIT\_VARIABLES= {

        "number\_of\_nums": "int"

    }

    def \_\_init\_\_(self, number\_generator\_cls:question.numGenType, number\_of\_nums:int=2) -> None:

        super().\_\_init\_\_()

        self.number\_generator\_obj = number\_generator\_cls

        if number\_of\_nums < 2: raise Exception("number\_of\_nums must be 2 or greater")

        self.number\_of\_nums = number\_of\_nums

    def generate\_question(self) -> question.Question:

        format\_string = "Find Profit Percentage When Cost Price is {} and Sell Price is {}"

        num\_list = [self.number\_generator\_obj.number() for \_ in range(2)]

        num\_list.sort()

        cp, sp = num\_list

        percentage = self.percent(cp, sp)

        question\_string = format\_string.format(cp, sp)

        return question.Question(question\_string, percentage, self.Q\_TYPE)

    def percent(self, cp, sp):

        return round(((sp - cp) / sp) \* 100, 2)

class ProfitQuestionType2(question.QuestionType):

    Q\_TYPE = "ProfitType2"

    INIT\_VARIABLES= {

        "number\_of\_nums": "int"

    }

    def \_\_init\_\_(self, number\_generator\_cls:question.numGenType, number\_of\_nums:int=2) -> None:

        super().\_\_init\_\_()

        self.number\_generator\_obj = number\_generator\_cls

        if number\_of\_nums < 2: raise Exception("number\_of\_nums must be 2 or greater")

        self.number\_of\_nums = number\_of\_nums

    def generate\_question(self) -> question.Question:

        format\_string = "Find Cost Price Percentage When Profit Percentage is {} and Sell Price is {}"

        num\_list = [self.number\_generator\_obj.number() for \_ in range(2)]

        num\_list.sort()

        cp, sp = num\_list

        percentage = self.percent(cp, sp)

        question\_string = format\_string.format(percentage, sp)

        return question.Question(question\_string, cp, self.Q\_TYPE)

    def percent(self, cp, sp):

        return round(((sp - cp) / sp) \* 100, 2)

class ProfitQuestionType3(question.QuestionType):

    Q\_TYPE = "ProfitType3"

    INIT\_VARIABLES= {

        "number\_of\_nums": "int"

    }

    def \_\_init\_\_(self, number\_generator\_cls:question.numGenType, number\_of\_nums:int=2) -> None:

        super().\_\_init\_\_()

        self.number\_generator\_obj = number\_generator\_cls

        if number\_of\_nums < 2: raise Exception("number\_of\_nums must be 2 or greater")

        self.number\_of\_nums = number\_of\_nums

    def generate\_question(self) -> question.Question:

        format\_string = "Find Sell Price Percentage When Profit Percentage is {} and Cost Price is {}"

        num\_list = [self.number\_generator\_obj.number() for \_ in range(2)]

        num\_list.sort()

        cp, sp = num\_list

        percentage = self.percent(cp, sp)

        question\_string = format\_string.format(percentage, cp)

        return question.Question(question\_string, sp, self.Q\_TYPE)

    def percent(self, cp, sp):

        return round(((sp - cp) / sp) \* 100, 2)

TYPE\_LOOKUP:dict[str, Type[question.QuestionType]] = {

    "Find Profit": ProfitQuestionType1,

    "FInd Cost": ProfitQuestionType2,

    "Find Sell": ProfitQuestionType3

}

**quadratic.py**

# question\_strategies/quadratic.py

# In-built imports

import math

from typing import Type

# Third-party imports

# Sys-Paths for Relative Imports

import sys

from os.path import dirname, abspath

package\_path = dirname(dirname(abspath(\_\_file\_\_)))

if(package\_path not in sys.path): sys.path.insert(0, package\_path)

# Relative imports

from question\_strategies import question

class QuadraticQuestionType1(question.QuestionType):

    Q\_TYPE = "Quadratic\_Type1"

    INIT\_VARIABLES= {}

    def \_\_init\_\_(self, number\_generator\_cls:question.numGenType) -> None:

        super().\_\_init\_\_()

        self.number\_generator\_obj = number\_generator\_cls

    def generate\_question(self):

        format\_string = "{}x^2 - {}x + {} = 0"

        roots = [self.number\_generator\_obj.number() for \_ in range(2)]

        coefficient = [1, sum(roots), math.prod(roots)]

        question\_string = format\_string.format(\*coefficient)

        return question.Question(question\_string, roots, self.Q\_TYPE.title())

TYPE\_LOOKUP:dict[str, Type[question.QuestionType]] = {

    "normal": QuadraticQuestionType1

}

**square.py**

# question\_strategies/addition.py

# In-built imports

from typing import Type

# Third-party imports

# Sys-Paths for Relative Imports

import sys

from os.path import dirname, abspath

package\_path = dirname(dirname(abspath(\_\_file\_\_)))

if(package\_path not in sys.path): sys.path.insert(0, package\_path)

# Relative imports

from question\_strategies import question

class SquareQuestionType1(question.QuestionType):

    Q\_TYPE = "SquareType1"

    INIT\_VARIABLES= {

        "number\_of\_nums": "int"

    }

    def \_\_init\_\_(self, number\_generator\_cls:question.numGenType, number\_of\_nums:int=1) -> None:

        super().\_\_init\_\_()

        self.number\_generator\_obj = number\_generator\_cls

        self.operator = "+"

        if number\_of\_nums < 1: raise Exception("number\_of\_nums must be 2 or greater")

        self.number\_of\_nums = number\_of\_nums

    def generate\_question(self) -> question.Question:

        format\_string = f"Find Square of: " + f"{self.operator}".join(["{}" for \_ in range(self.number\_of\_nums)])

        num\_list = [self.number\_generator\_obj.number() for \_ in range(self.number\_of\_nums)]

        question\_string = format\_string.format(\*num\_list)

        return question.Question(question\_string, self.sq(num\_list)[0], self.Q\_TYPE)

    def sq(self, num\_list):

        return [i\*i for i in num\_list]

TYPE\_LOOKUP:dict[str, Type[question.QuestionType]] = {

    "normal": SquareQuestionType1

}

**subtraction.py**

# question\_strategies/subtraction.py

# In-built imports

import random

from typing import Type, Union

# Third-party imports

# Sys-Paths for Relative Imports

import sys

from os.path import dirname, abspath

package\_path = dirname(dirname(abspath(\_\_file\_\_)))

if(package\_path not in sys.path): sys.path.insert(0, package\_path)

# Relative imports

from question\_strategies import question

class SubtractionQuestionType1(question.QuestionType):

    Q\_TYPE = "Subtraction\_Type1"

    INIT\_VARIABLES= {

        "number\_of\_nums": "int"

    }

    def \_\_init\_\_(self, number\_generator\_cls:question.numGenType, number\_of\_nums:int=2) -> None:

        super().\_\_init\_\_()

        self.number\_generator\_obj = number\_generator\_cls

        self.operator = "-"

        if number\_of\_nums < 2: raise Exception("number\_of\_nums must be 2 or greater")

        self.number\_of\_nums = number\_of\_nums

    def generate\_question(self):

        format\_string = f" {self.operator} ".join(["{}" for \_ in range(self.number\_of\_nums)])

        num\_list = [self.number\_generator\_obj.number() for \_ in range(self.number\_of\_nums)]

        first\_num = sum(num\_list) + self.number\_generator\_obj.number()

        question\_string = format\_string.format(first\_num, \*num\_list)

        return question.Question(question\_string, eval(question\_string), self.Q\_TYPE)

class SubtractionQuestionType2(question.QuestionType):

    Q\_TYPE = "Subtraction\_Type2"

    INIT\_VARIABLES= {

        "number\_of\_nums": "int"

    }

    def \_\_init\_\_(self, number\_generator\_cls:question.numGenType, number\_of\_nums:int=2) -> None:

        super().\_\_init\_\_()

        self.number\_generator\_obj = number\_generator\_cls

        self.operator = "-"

        if number\_of\_nums < 2: raise Exception("number\_of\_nums must be 2 or greater")

        self.number\_of\_nums = number\_of\_nums

    def generate\_question(self):

        format\_string = f" {self.operator} ".join(["{}" for \_ in range(self.number\_of\_nums)])

        num\_list = [self.number\_generator\_obj.number() for \_ in range(self.number\_of\_nums)]

        first\_num = sum(num\_list) + self.number\_generator\_obj.number(is\_negative=True)

        question\_string = format\_string.format(first\_num, \*num\_list)

        return question.Question(question\_string, eval(question\_string), self.Q\_TYPE)

class SubtractionQuestionType3(question.QuestionType):

    Q\_TYPE = "Subtraction\_Type3"

    INIT\_VARIABLES= {

        "number\_of\_nums": "int"

    }

    def \_\_init\_\_(self, number\_generator\_cls:question.numGenType, number\_of\_nums:int=2) -> None:

        super().\_\_init\_\_()

        self.number\_generator\_obj = number\_generator\_cls

        self.operator = "-"

        if number\_of\_nums < 2: raise Exception("number\_of\_nums must be 2 or greater")

        if (number\_of\_nums % 2) != 0: raise Exception("number\_of\_nums must be even")

        self.number\_of\_nums = number\_of\_nums

    def generate\_question(self):

        format\_string = f" {self.operator} ".join(["{}" for \_ in range(self.number\_of\_nums)])

        num\_list = [self.number\_generator\_obj.number() for \_ in range(self.number\_of\_nums//2)]

        num\_list += [-n for n in num\_list]

        random.shuffle(num\_list)

        question\_string = format\_string.format(\*num\_list)

        return question.Question(question\_string, eval(question\_string), self.Q\_TYPE.title())

TYPE\_LOOKUP:dict[str, Type[question.QuestionType]] = {

    "positive": SubtractionQuestionType1,

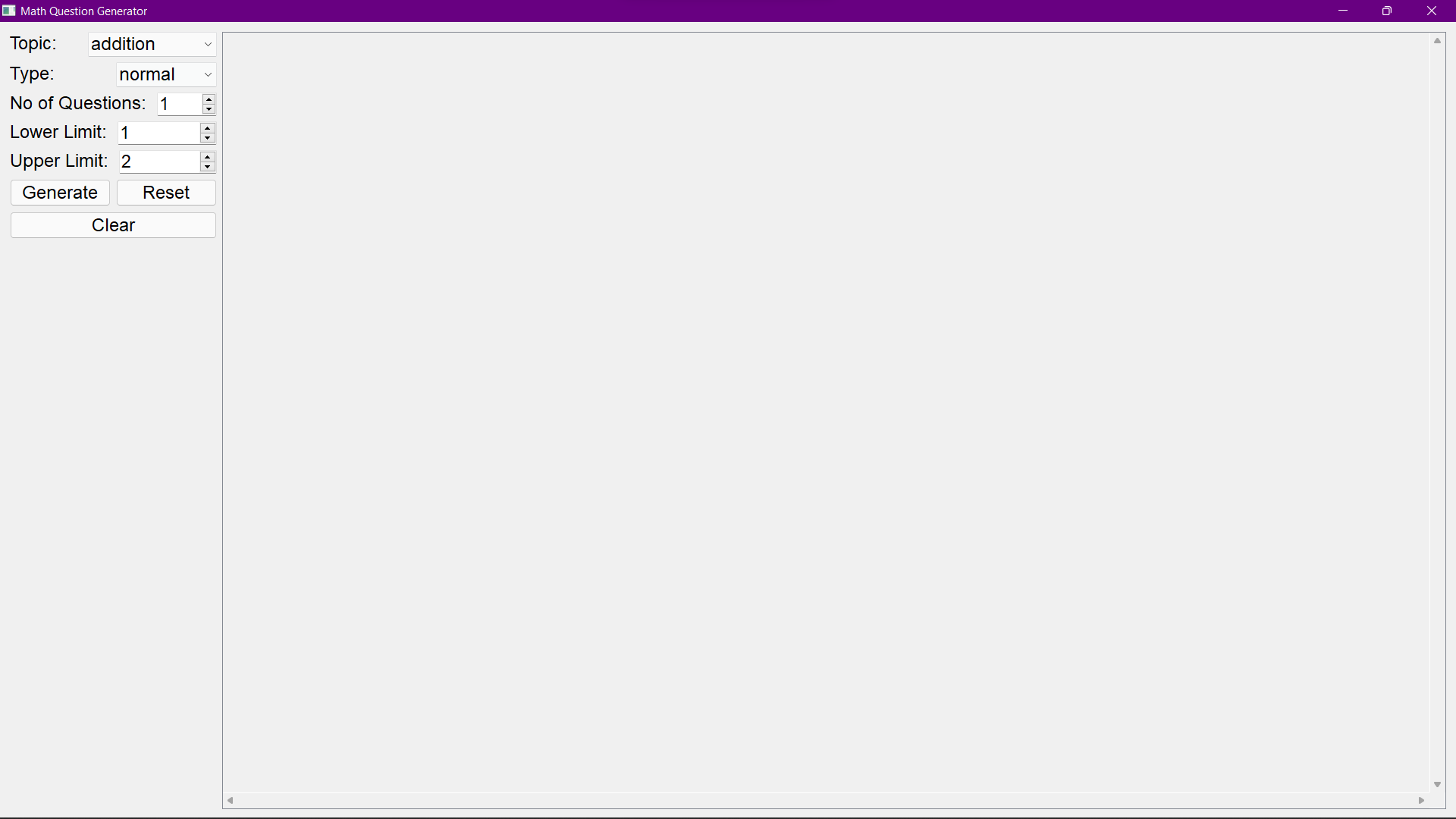
    "negative": SubtractionQuestionType2,

    "zeros": SubtractionQuestionType3

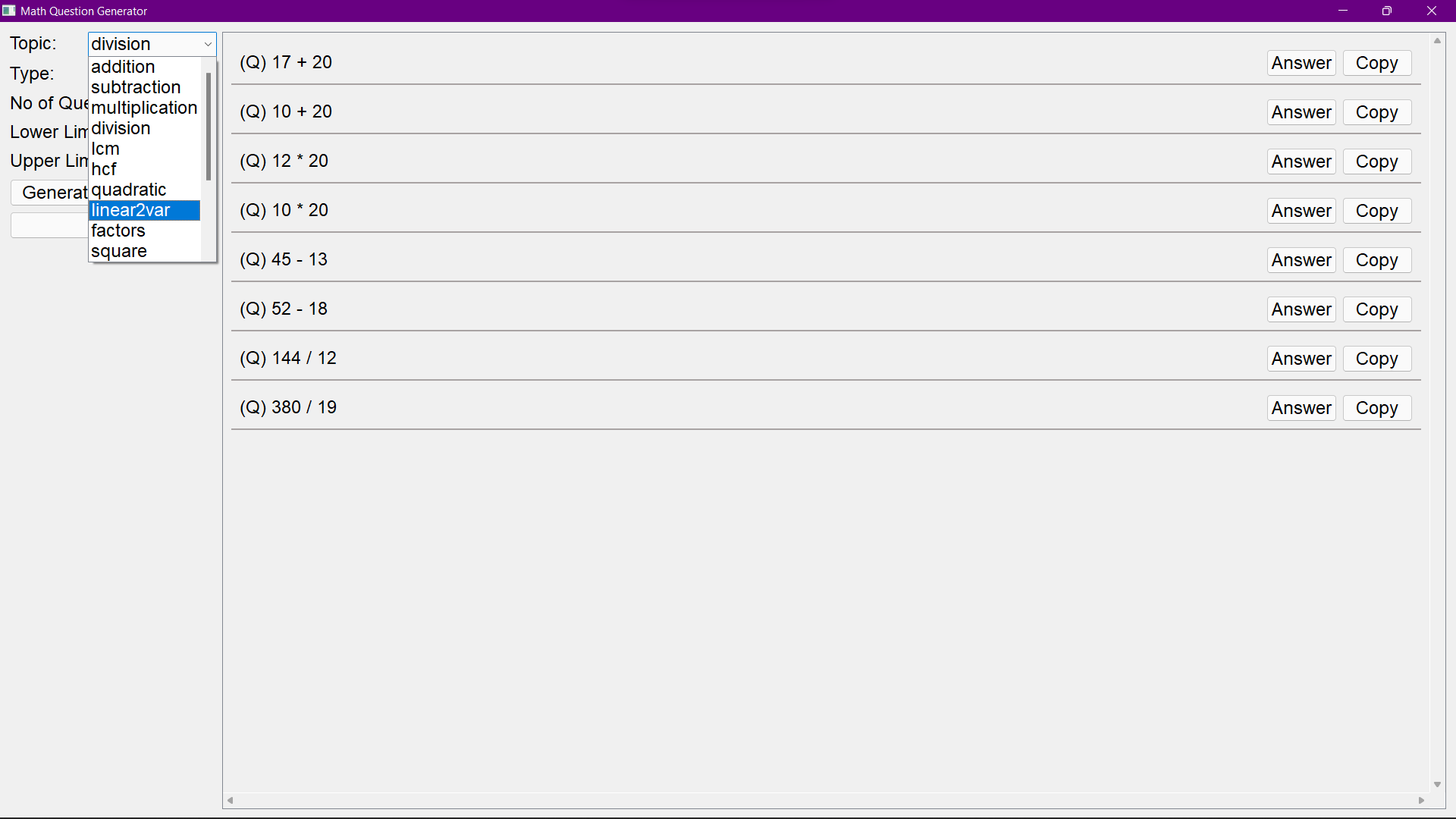
}

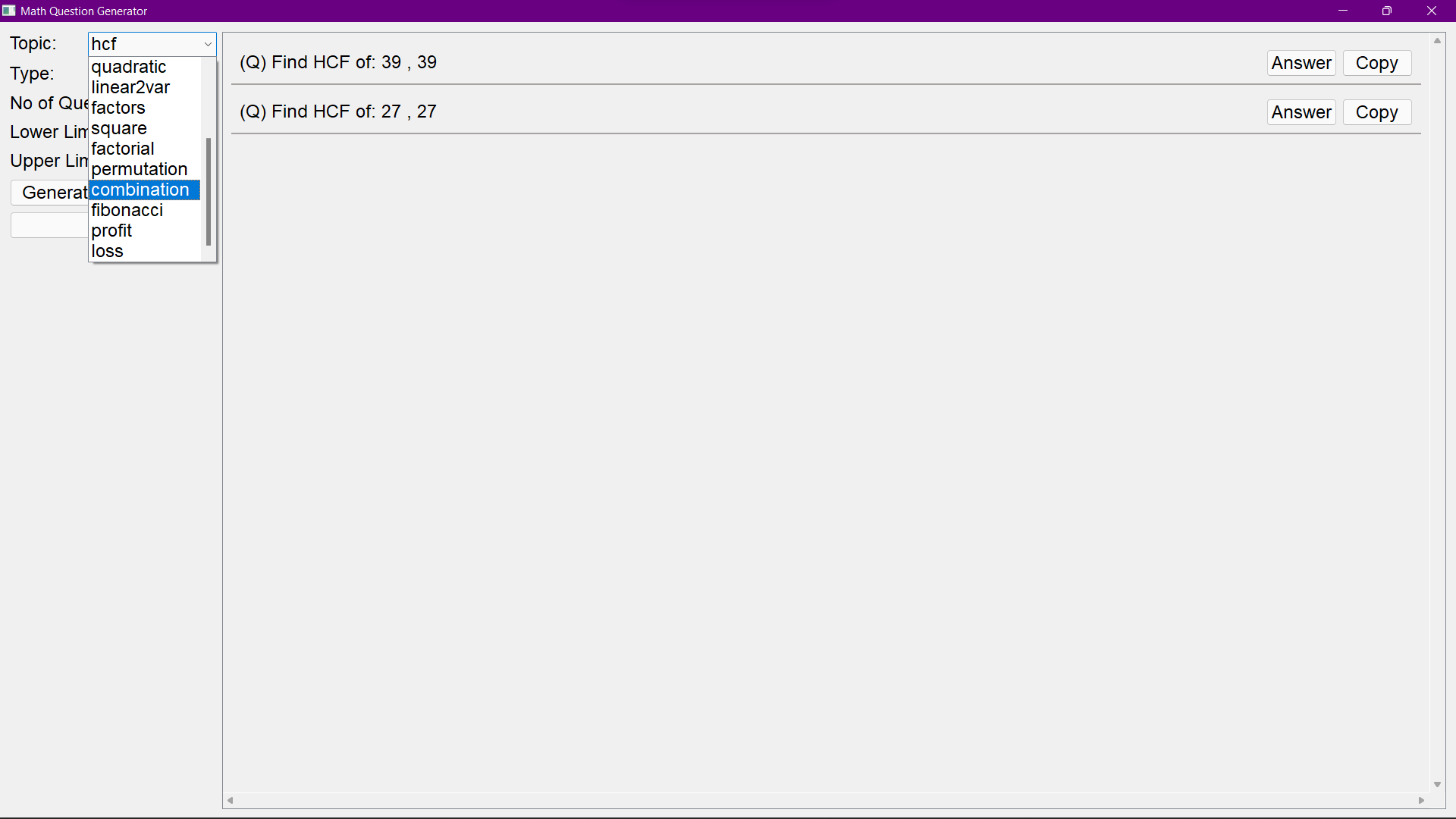
**Screen Layout**

* Start page

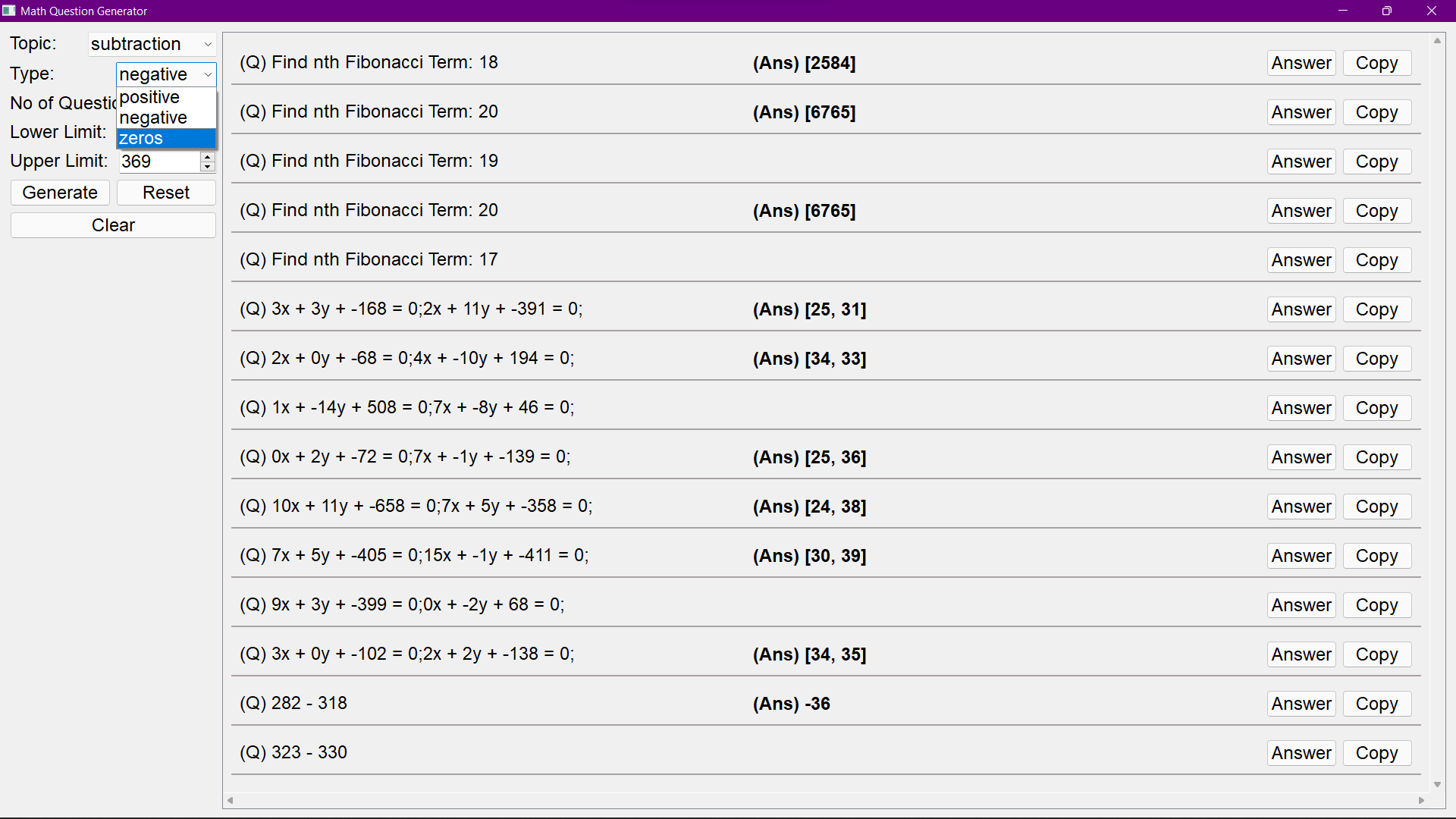


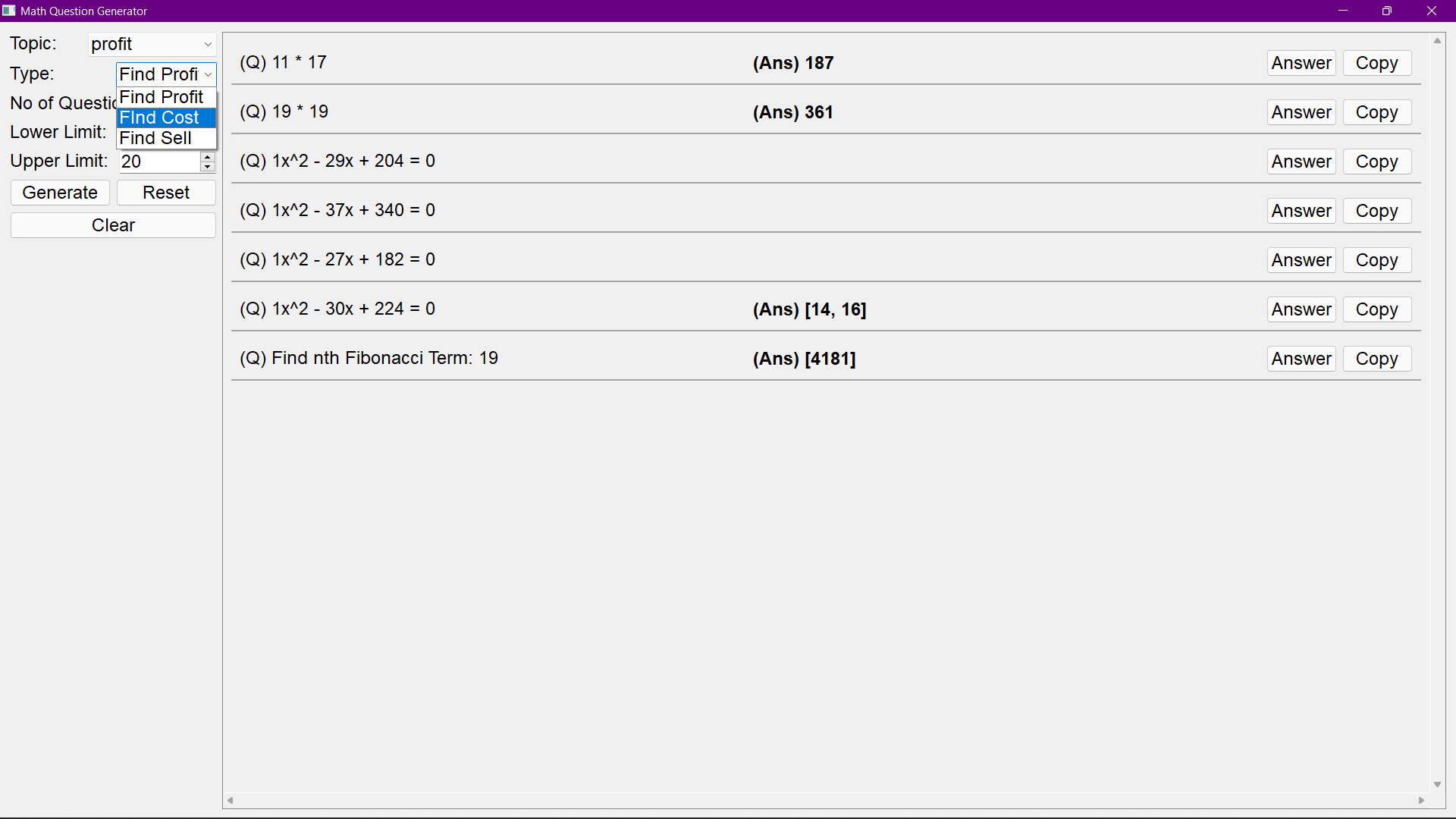
* Select Topic



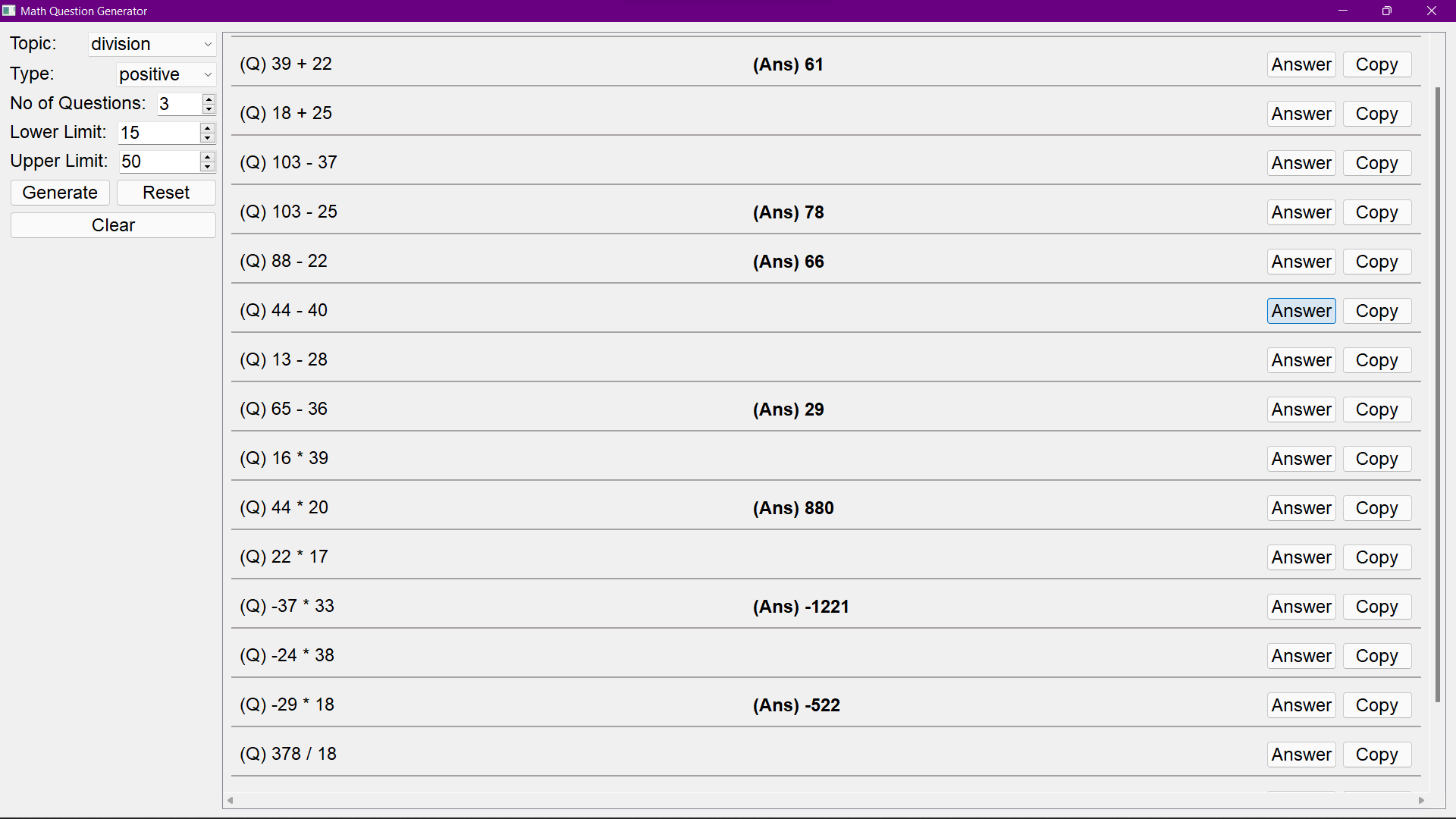


* Select Type

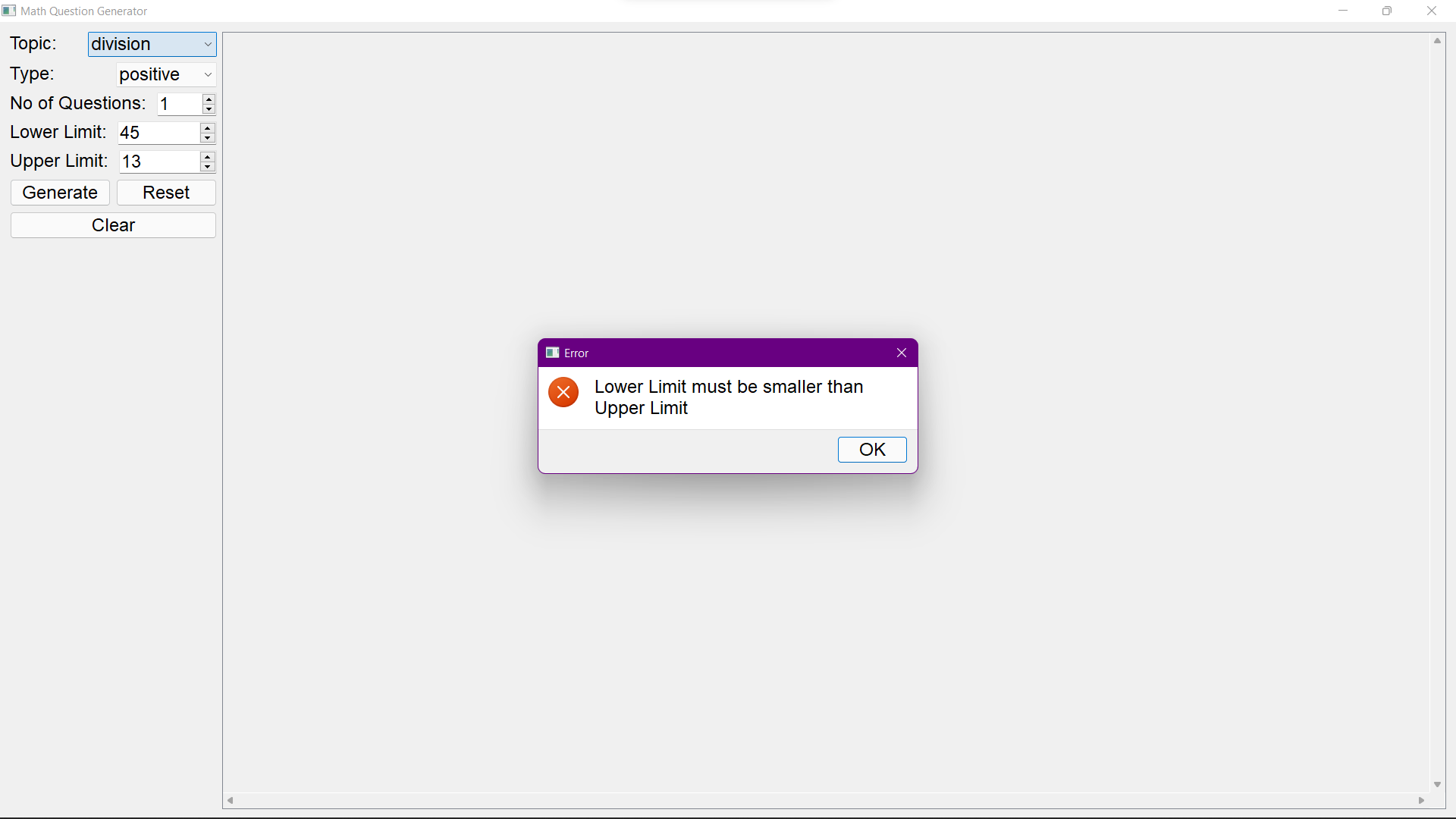




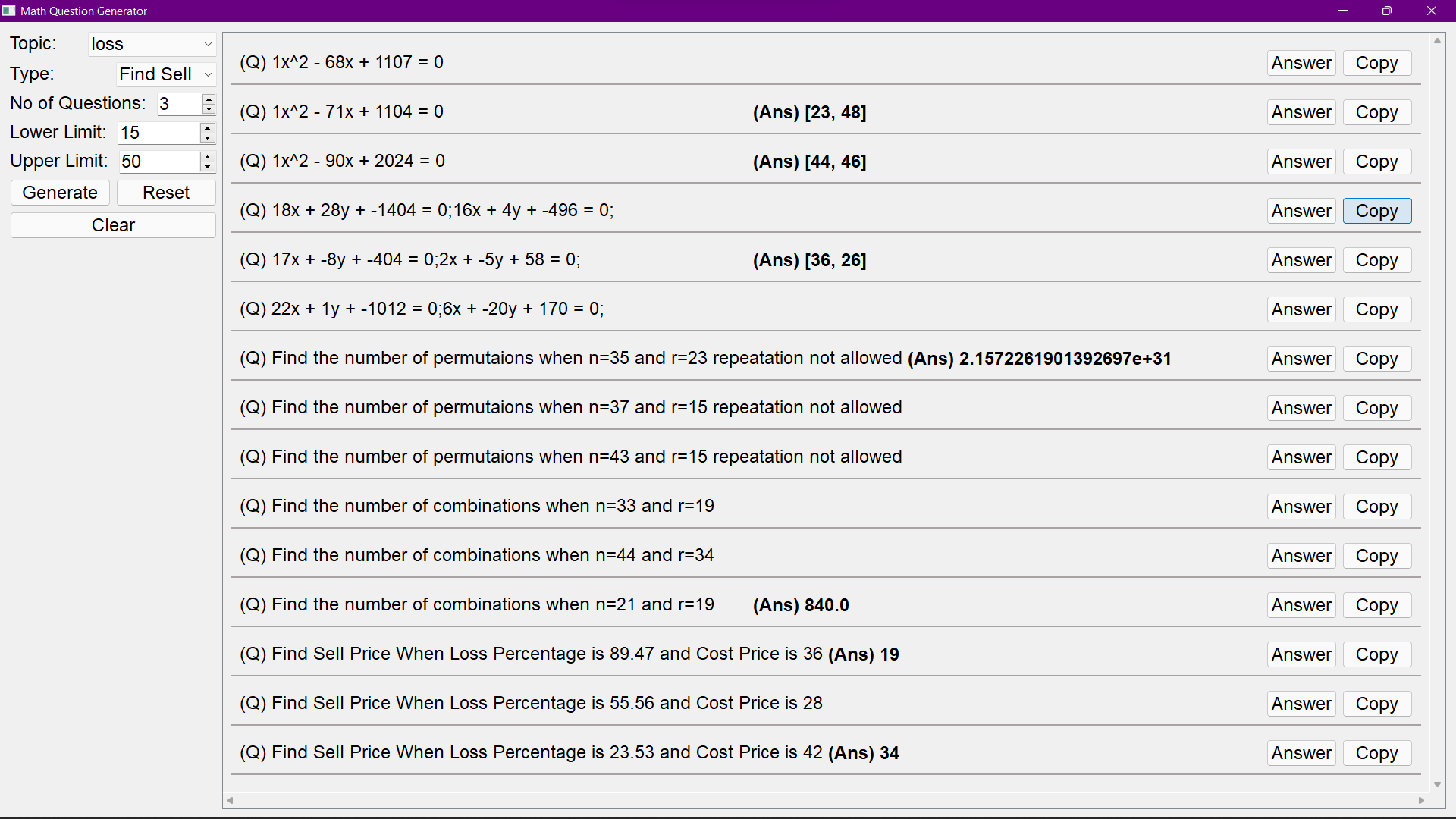
* Show Answer

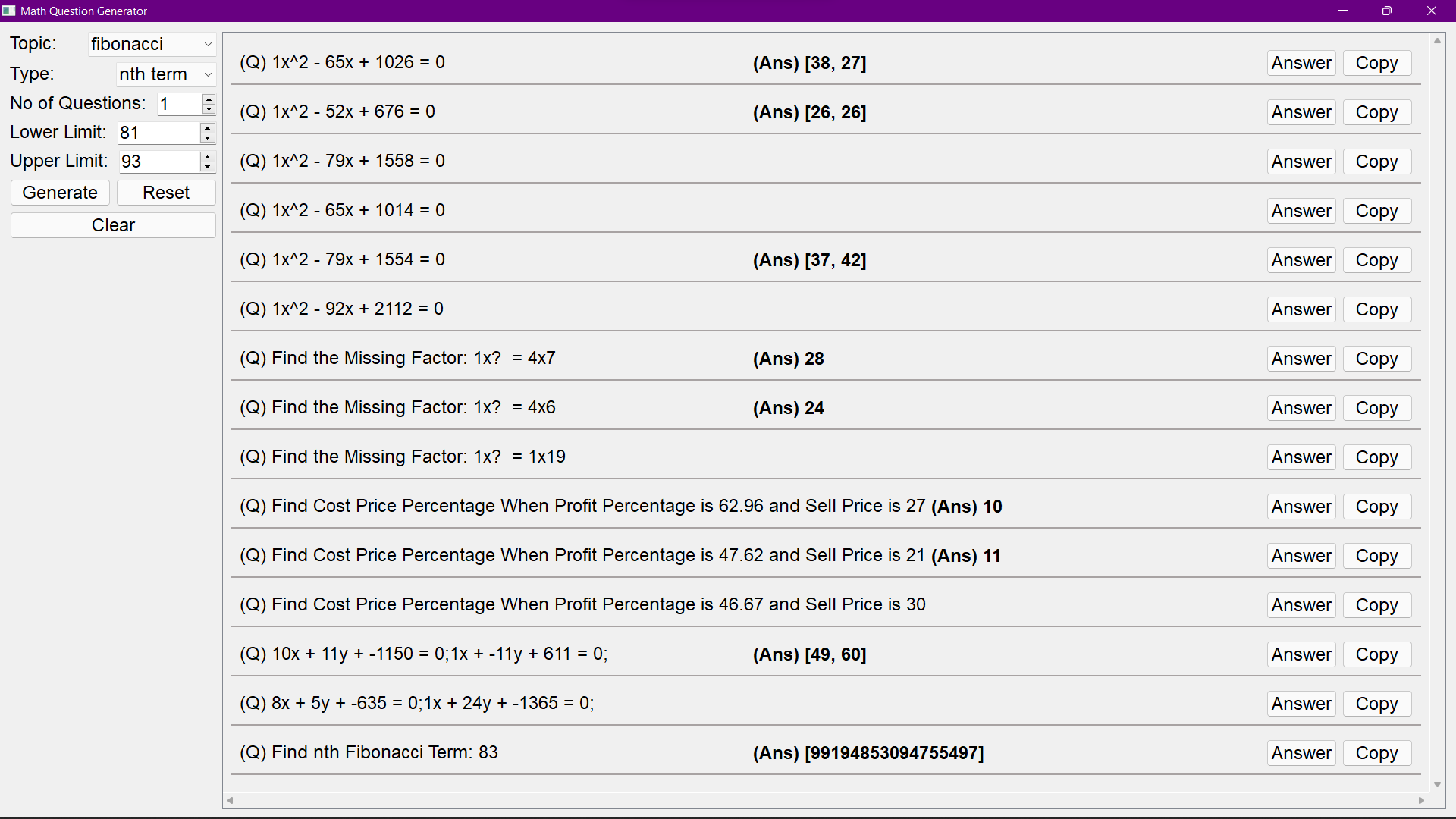


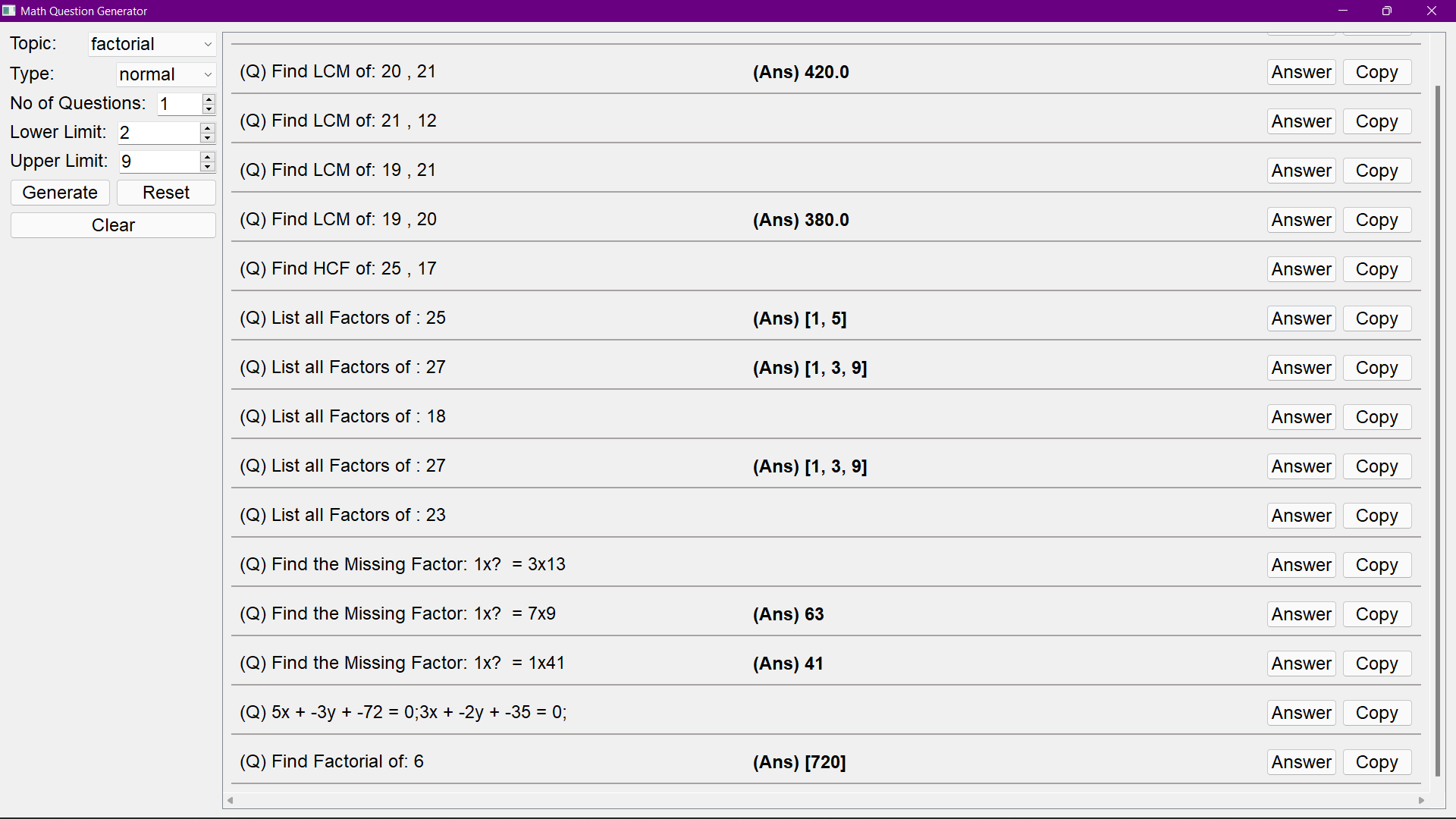
* Limit Error



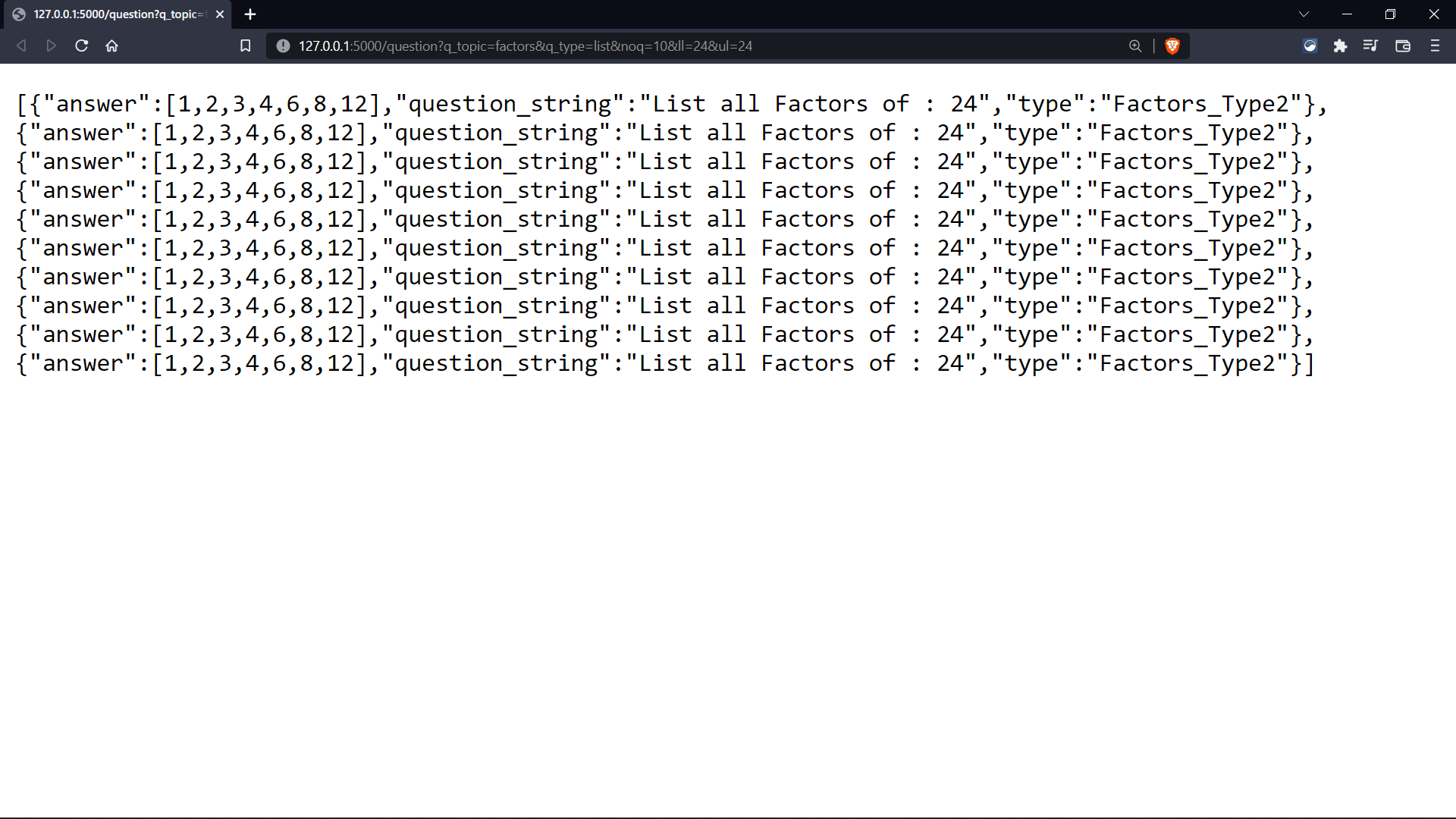
* Usage Examples

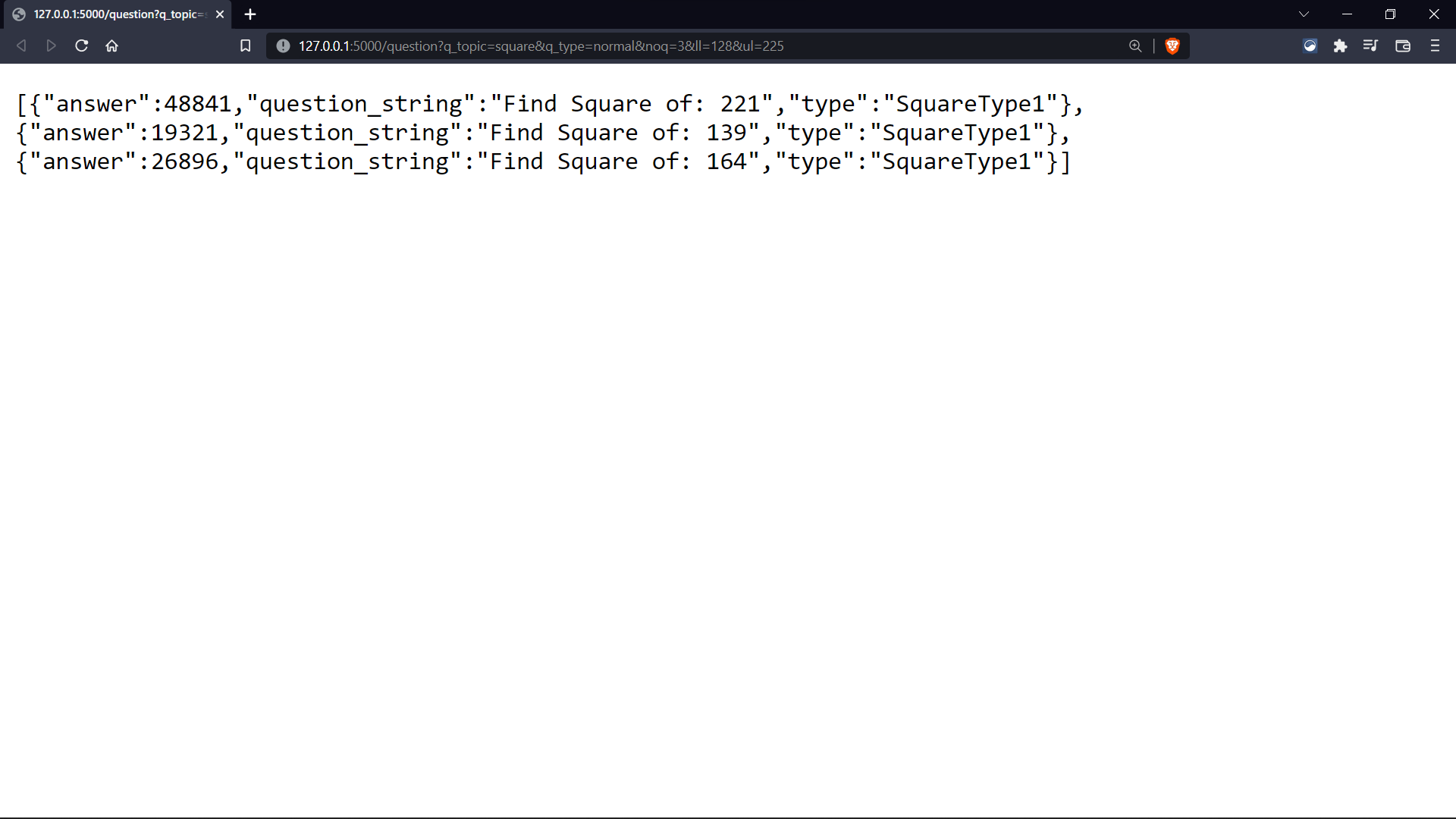


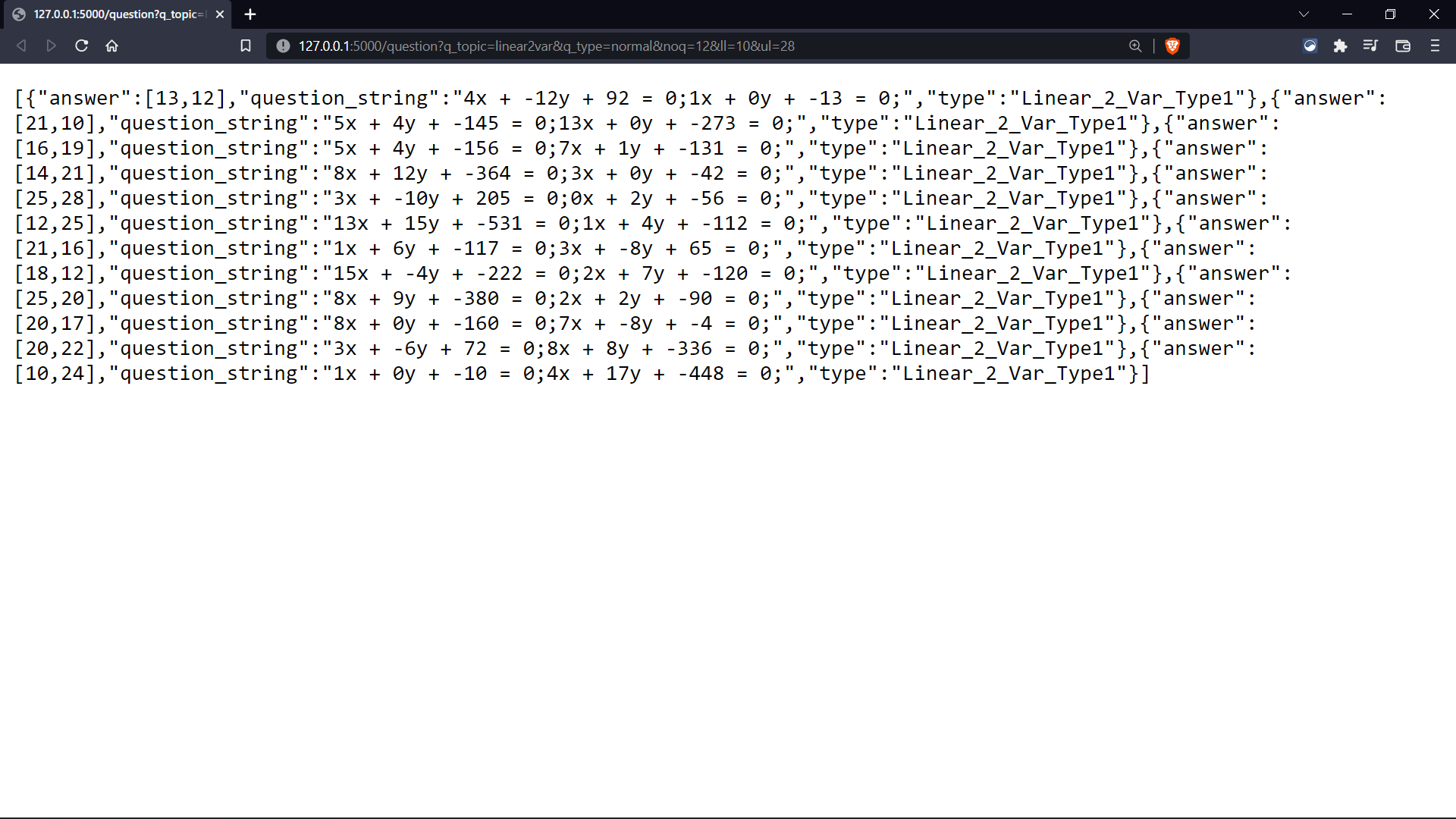




* Browser Example







**Future Enhancements**

* Re-arrange and remove questions from generated list of questions.
* Download PDF of generated questions.
* Question paper maker using generated questions.
* Online MCQ quiz platform using generated questions.
* Fraction numbers for question generation.
* More parameters for question generation to increase range of questions.
* Provide difficulty templates for different grade/ level of students.
* Provide step by step solution for any generated question.

**Bibliography**

Python – <https://www.python.org/>

Flask – <https://flask.palletsprojects.com/en/2.0.x/>

Elebetsamer – [math-worksheet-generator github repository](https://github.com/elebetsamer/math-worksheet-generator)

Lukew3 – [mathgenerator github repository](https://lukew3.github.io/mathgenerator/)

Januschung – [math-worksheet-generator github repository](https://github.com/januschung/math-worksheet-generator)

Teacher’s Corner – <https://worksheets.theteacherscorner.net/make-your-own/math-worksheets/basic-math/math.php>

Draw.io – <https://app.diagrams.net/>