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

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/**math-worksheet-generator**, lukew3/**mathgenerator** and, 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/**math-worksheet-generator**, Teacher's Corner, Wolramalpha and, mathsbot.com's question generator.

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 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

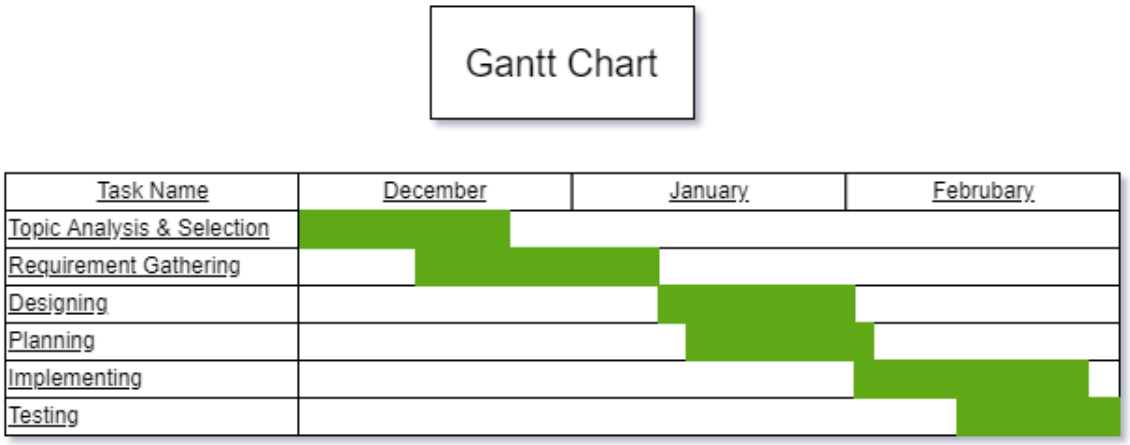
Individuals and organizations who are actively involved in the project, or whose interests may be positively or negatively affected as a result of project execution or successful project completion.

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

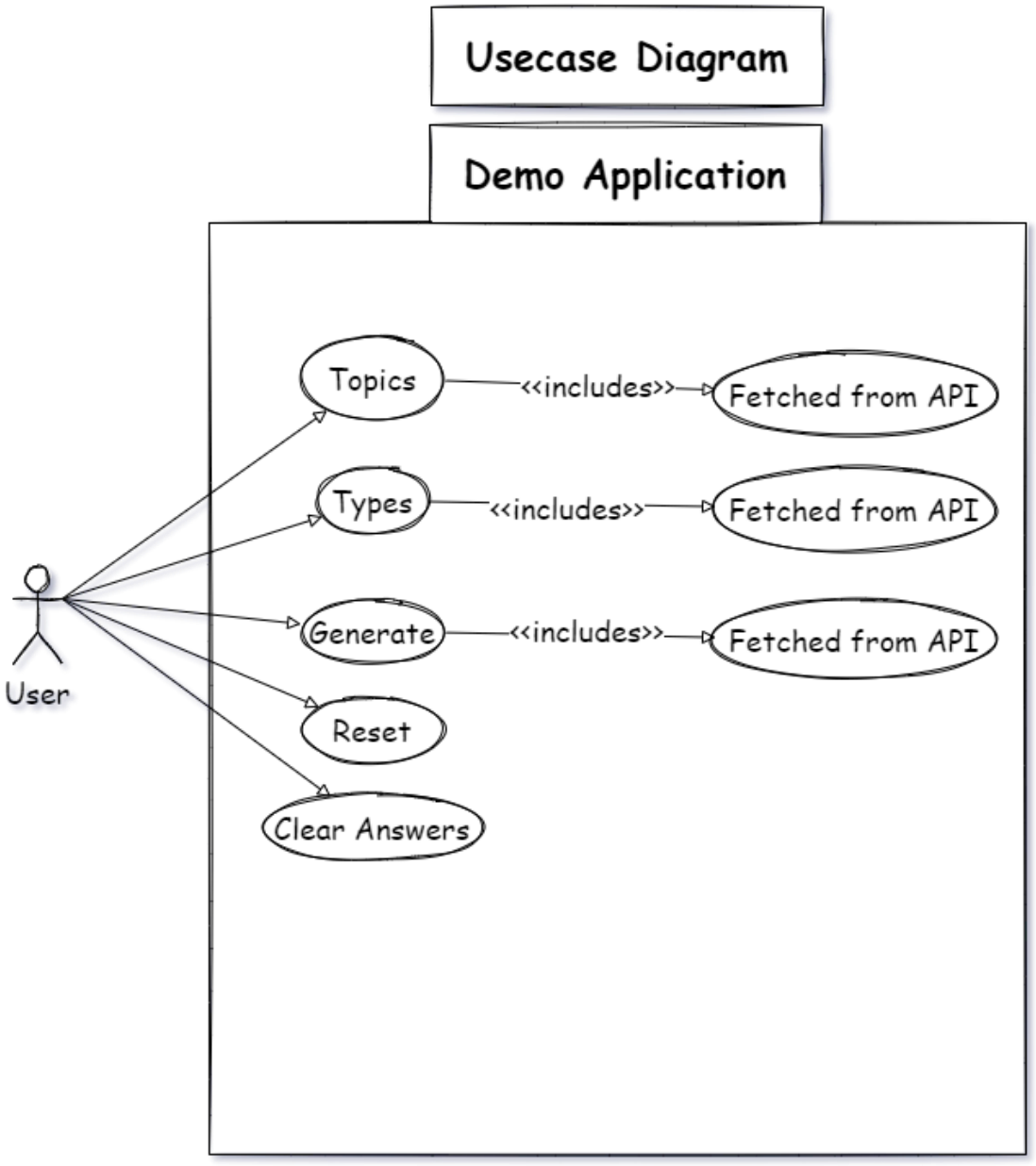
Event Table is a catalogue of use cases listed by event. Contains detailed information

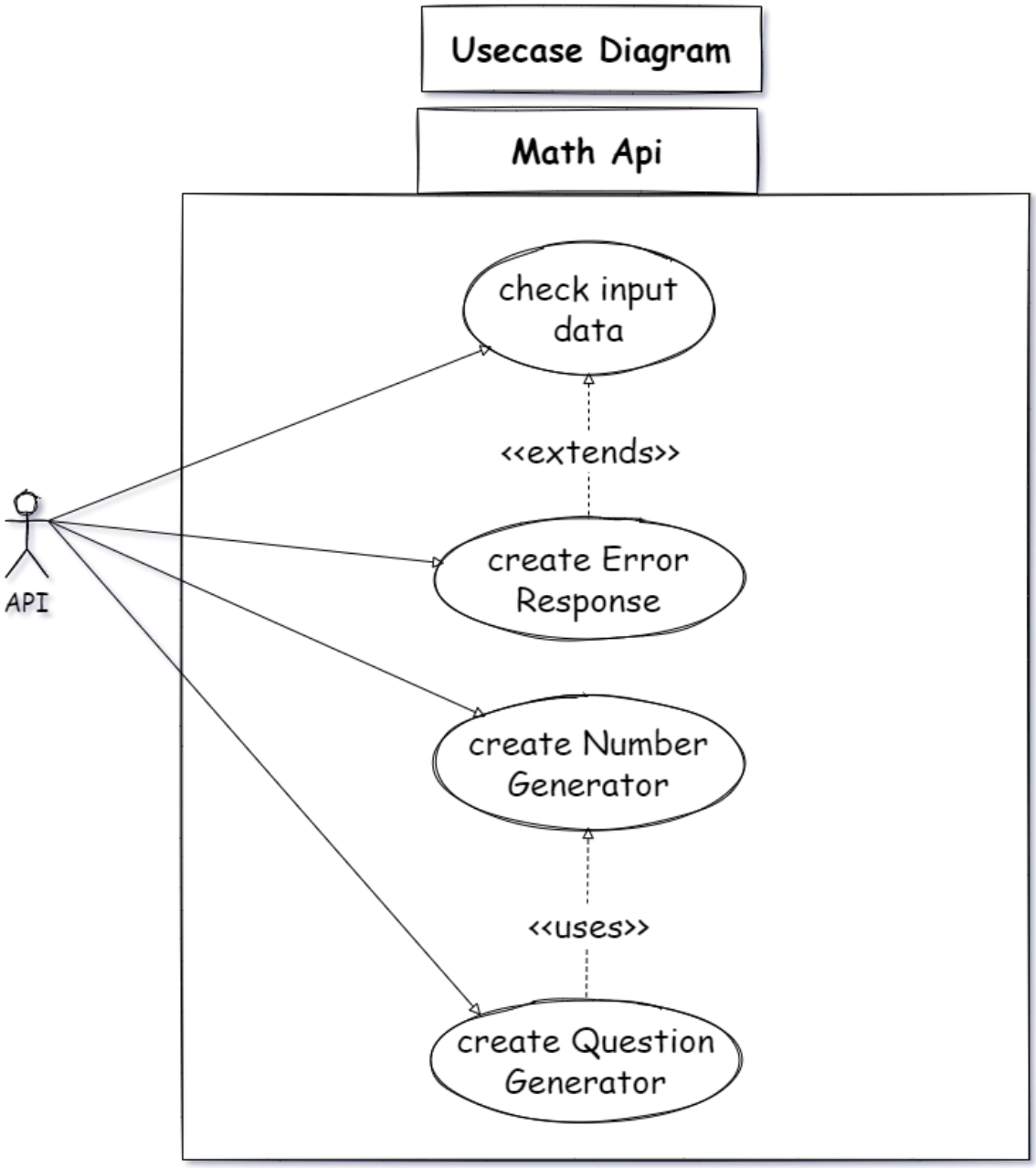
- Trigger: Signal that indicates an event has occurred.
- Source: External agent that initiates event and supplies data for the event.
- Response: Output produced by the system.
- Destination: External agent that receives the response.

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

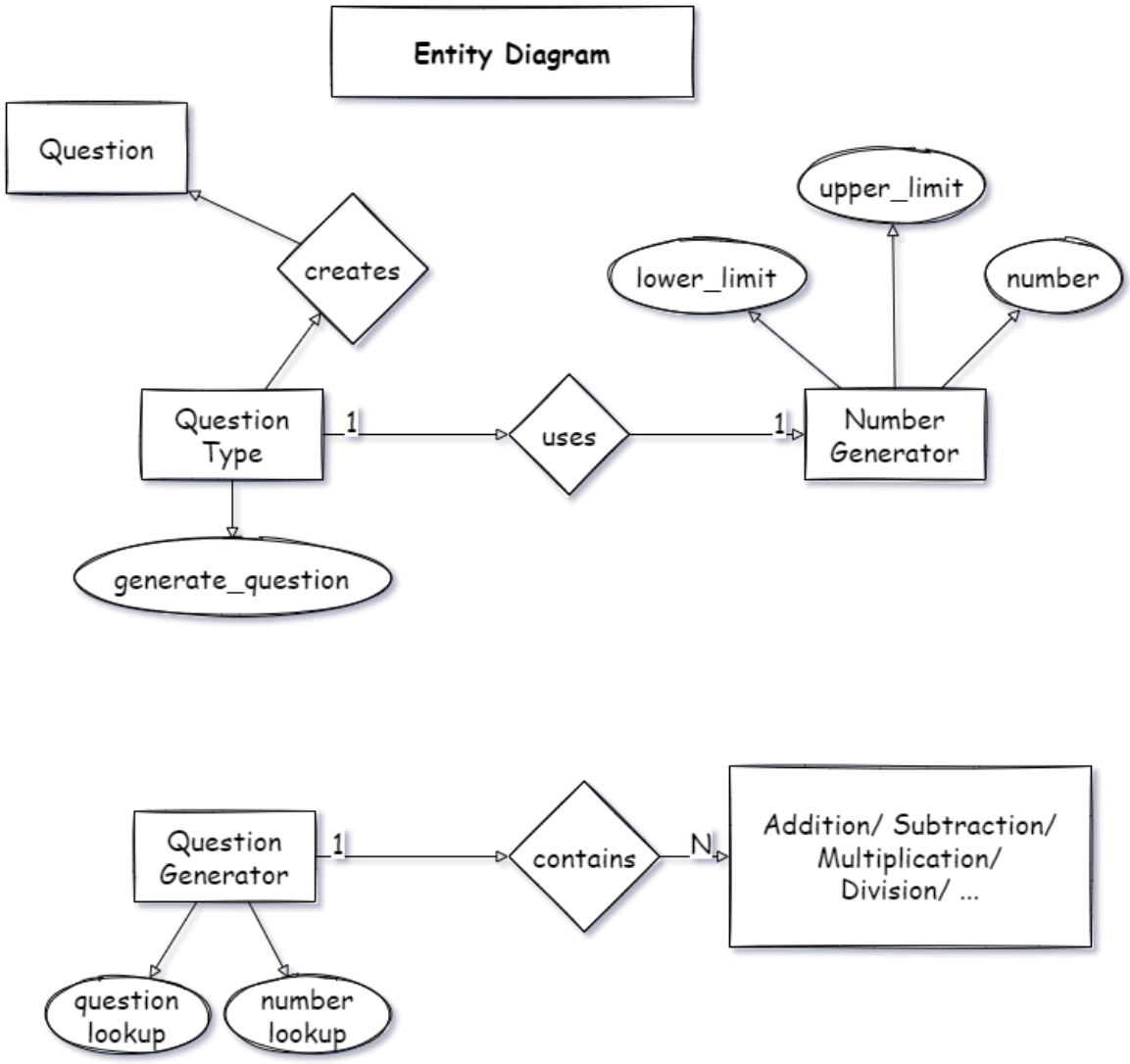
Use Case Diagrams are used to summarize the details of any system's users (also known as actors) and their interactions with the system.




Usecase Diagram
Math Api
check input data
<<extends>>
create Error Response
create Number Generator
<<uses>>
create Question Generator
API

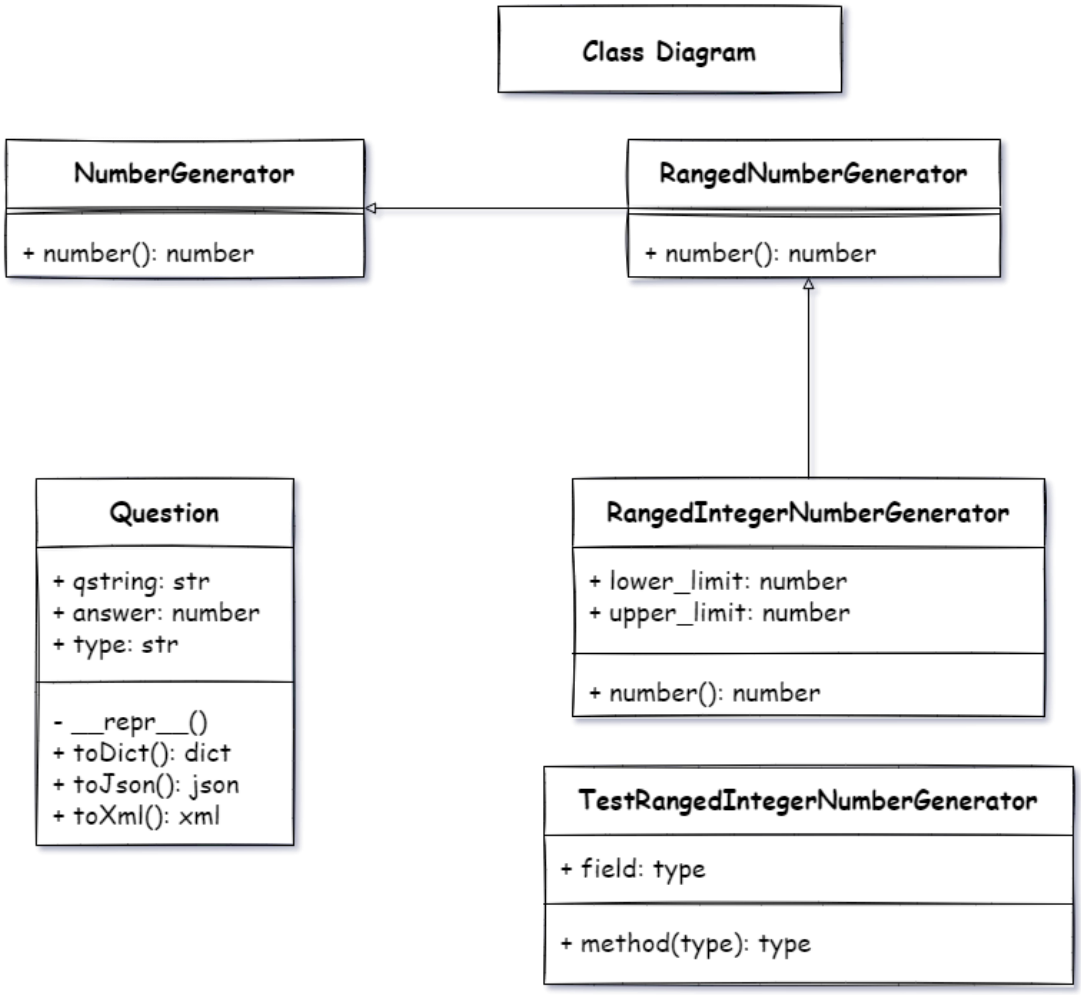
Entity-Relationship Diagram

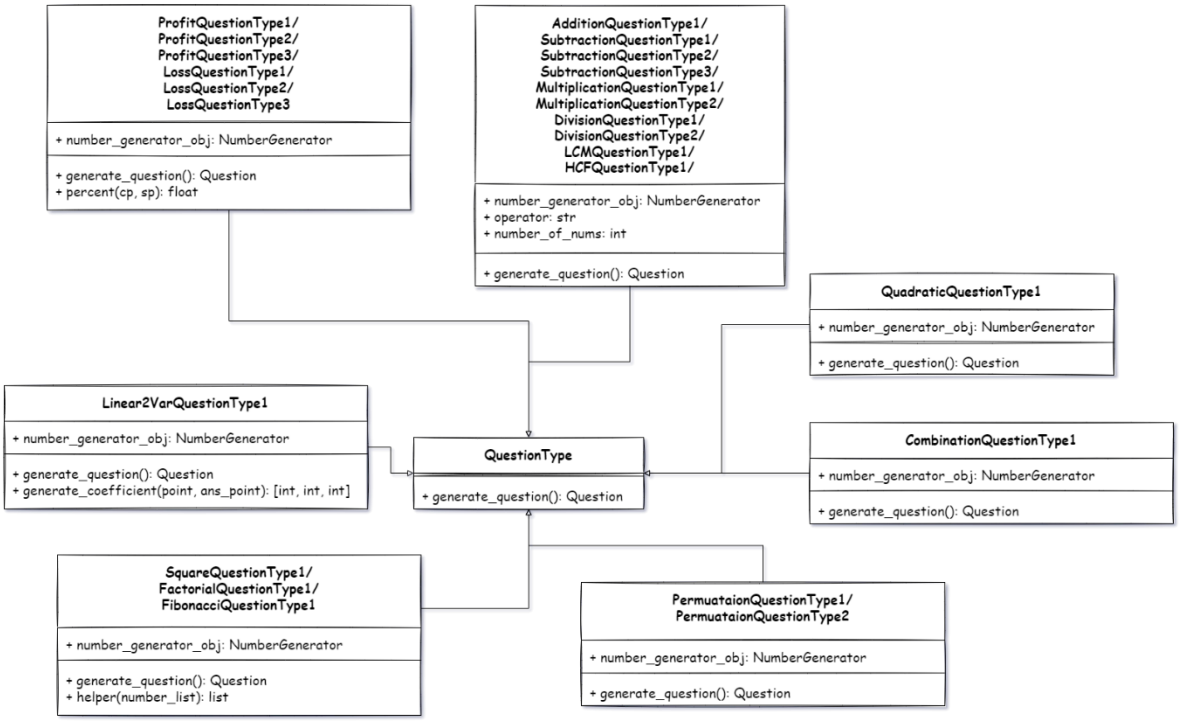
Entity Relationship Diagram (ERD) shows the relationships of entity sets stored in a database. An entity in this context is an object, a component of data. An entity set is a collection of similar entities. These entities can have attributes that define its properties.

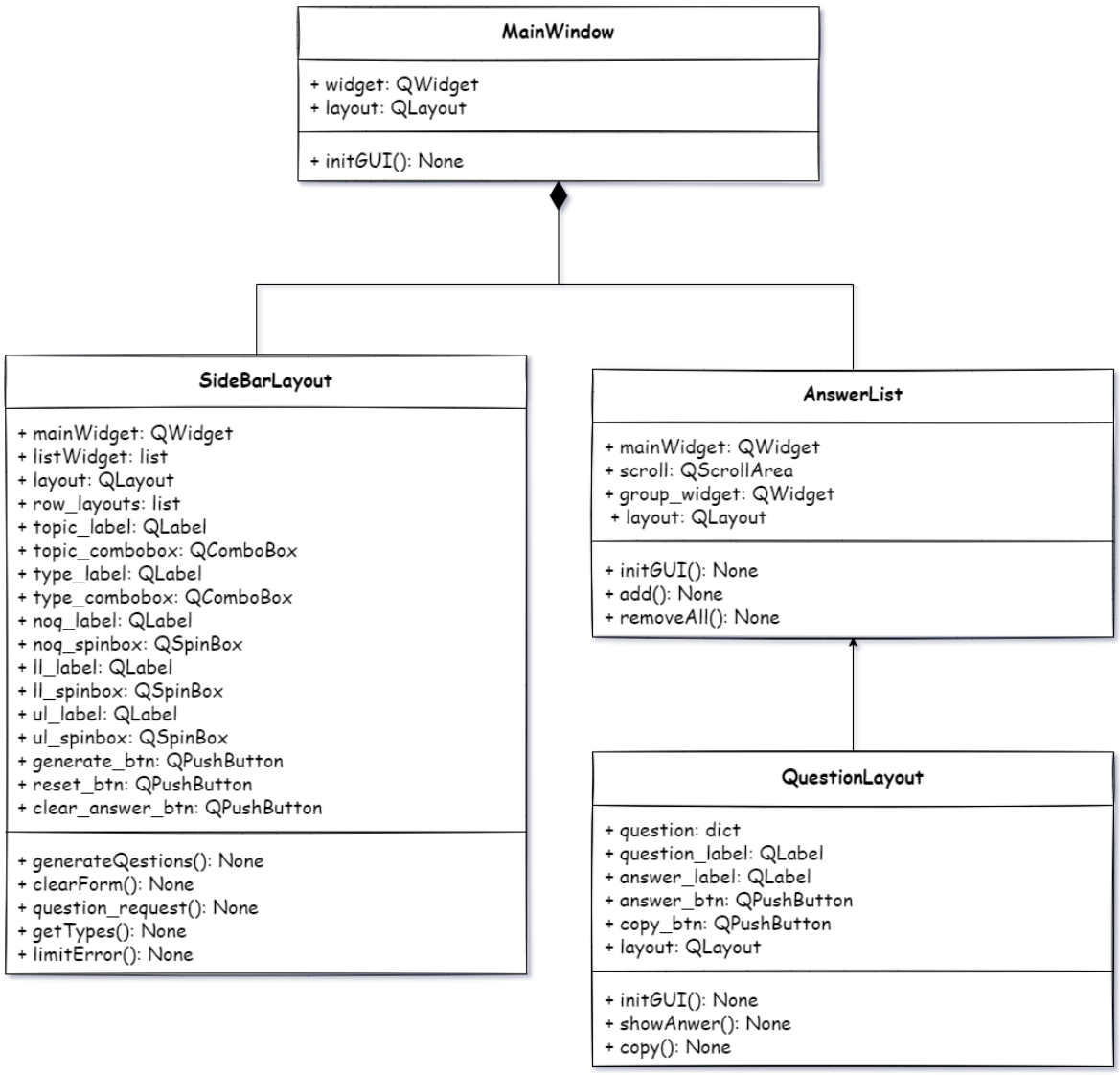


Class Diagram

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.

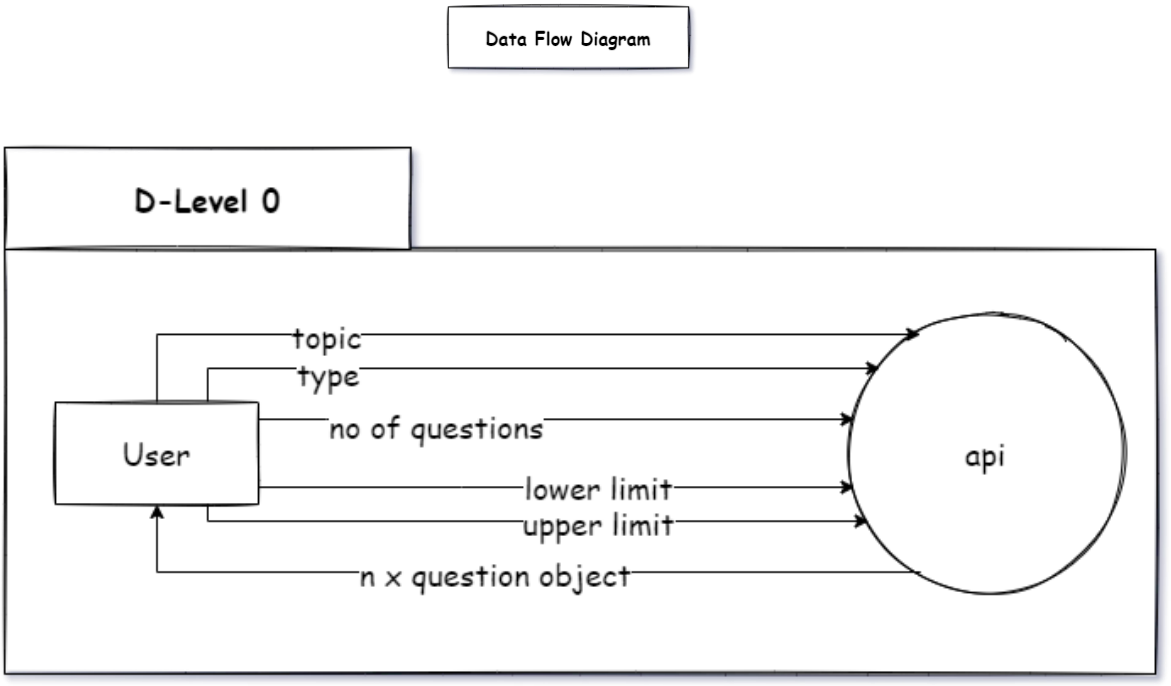


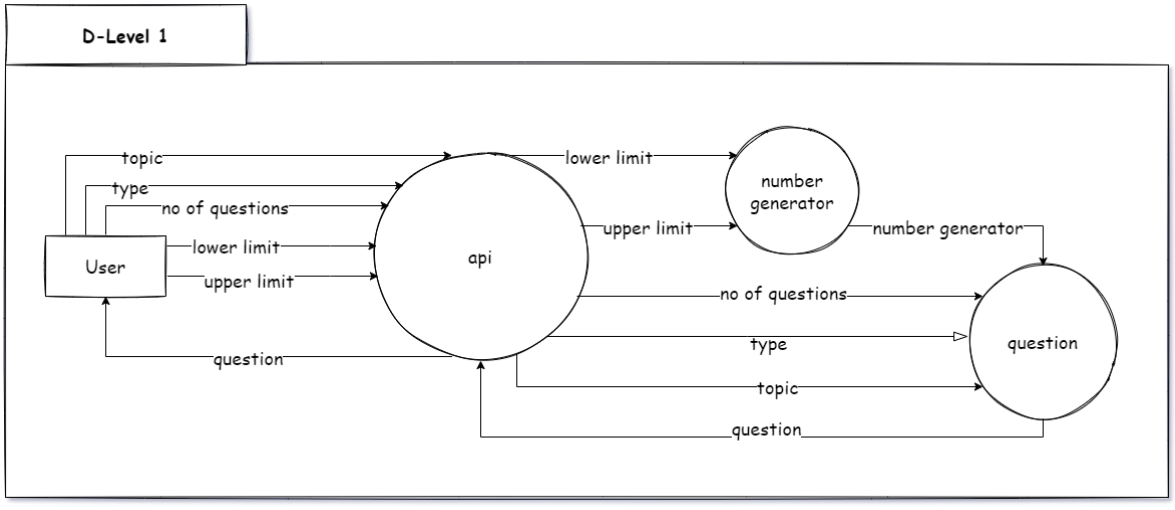

ProfitQuestionType1/
ProfitQuestionType2/
ProfitQuestionType3/
LossQuestionType1/
LossQuestionType2/
LossQuestionType3
+ number_generator_obj: NumberGenerator
+ generate_question(): Question
+ percent(cp, sp): float
AdditionQuestionType1/
SubtractionQuestionType1/
SubtractionQuestionType2/
SubtractionQuestionType3/
MultiplicationQuestionType1/
MultiplicationQuestionType2/
DivisionQuestionType1/
DivisionQuestionType2/
LCMQuestionType1/
HCFQuestionType1/
+ number_generator_obj: NumberGenerator
+ operator: str
+ number_of_nums: int
+ generate_question(): Question
QuadraticQuestionType1
+ number_generator_obj: NumberGenerator
+ generate_question(): Question
Linear2VarQuestionType1
+ number_generator_obj: NumberGenerator
+ generate_question(): Question
+ generate_coefficient(point, ans_point): [int, int, int]
QuestionType
+ generate_question(): Question
CombinationQuestionType1
+ number_generator_obj: NumberGenerator
+ generate_question(): Question
SquareQuestionType1/
FactorialQuestionType1/
FibonacciQuestionType1
+ number_generator_obj: NumberGenerator
+ generate_question(): Question
+ helper(number_list): list
PermuataionQuestionType1/
PermuataionQuestionType2
+ number_generator_obj: NumberGenerator
+ generate_question(): Question


MainWindow
+ widget: QWidget
+ layout: QLayout
+ initGUI(): None
SideBarLayout
+ mainWidget: QWidget
+ listWidget: list
+ layout: QLayout
+ row_layouts: list
+ topic_label: QLabel
+ topic_combobox: QComboBox
+ type_label: QLabel
+ type_combobox: QComboBox
+ noq_label: QLabel
+ noq_spinbox: QSpinBox
+ ll_label: QLabel
+ ll_spinbox: QSpinBox
+ ul_label: QLabel
+ ul_spinbox: QSpinBox
+ generate_btn: QPushButton
+ reset_btn: QPushButton
+ clear_answer_btn: QPushButton
+ generateQestions(): None
+ clearForm(): None
+ question_request(): None
+ getTypes(): None
+ limitError(): None
AnswerList
+ mainWidget: QWidget
+ scroll: QScrollArea
+ group_widget: QWidget
+ layout: QLayout
+ initGUI(): None
+ add(): None
+ removeAll(): None
QuestionLayout
+ question: dict
+ question_label: QLabel
+ answer_label: QLabel
+ answer_btn: QPushButton
+ copy_btn: QPushButton
+ layout: QLayout
+ initGUI(): None
+ showAnwer(): None
+ copy(): None

Data Flow Diagram

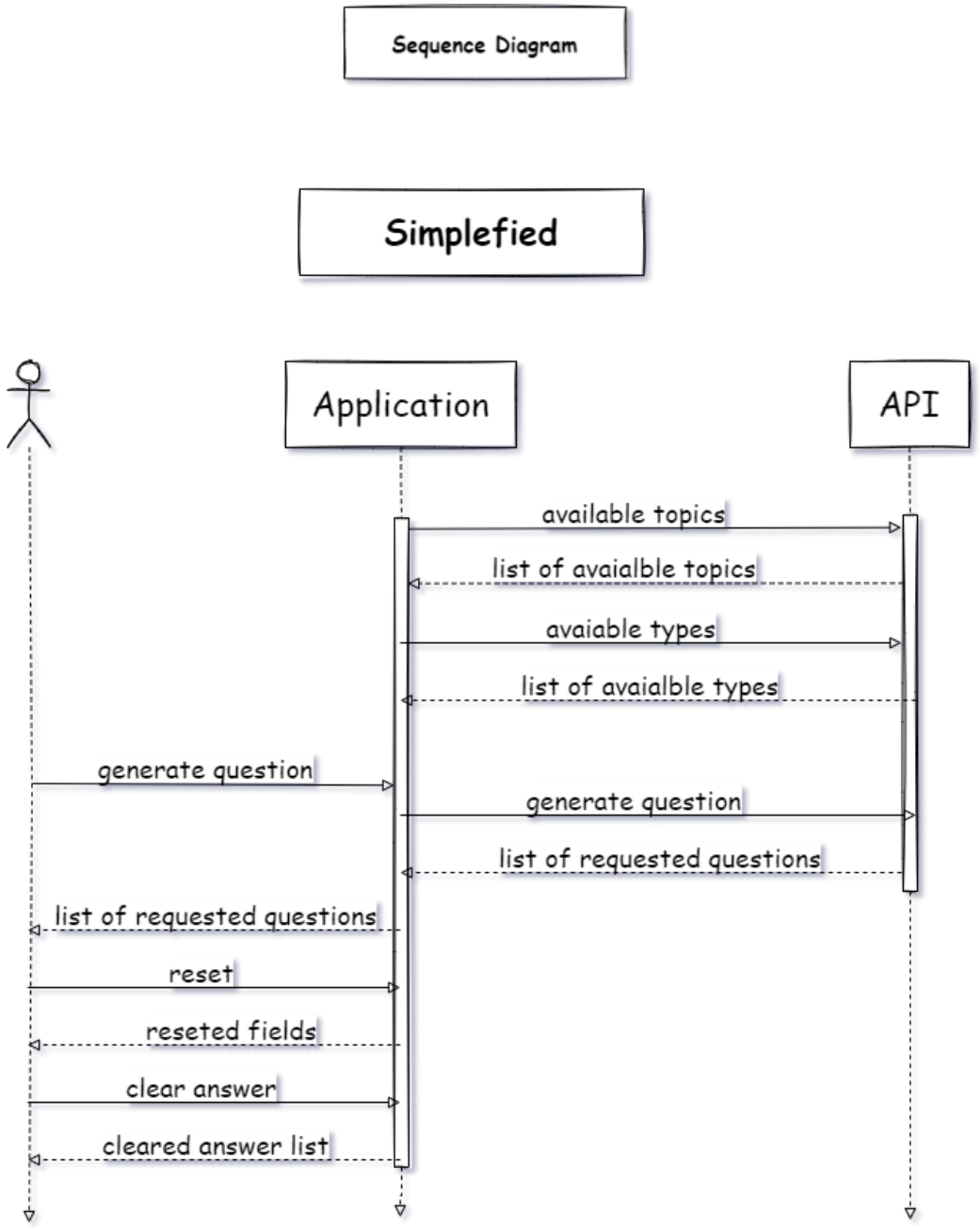
Data Flow Diagram (DFD) describes the in and out the flow of information for any process or system. It uses defined symbols like rectangles, circles and arrows, plus short text labels, to show data inputs, outputs, storage points and the routes between each destination.

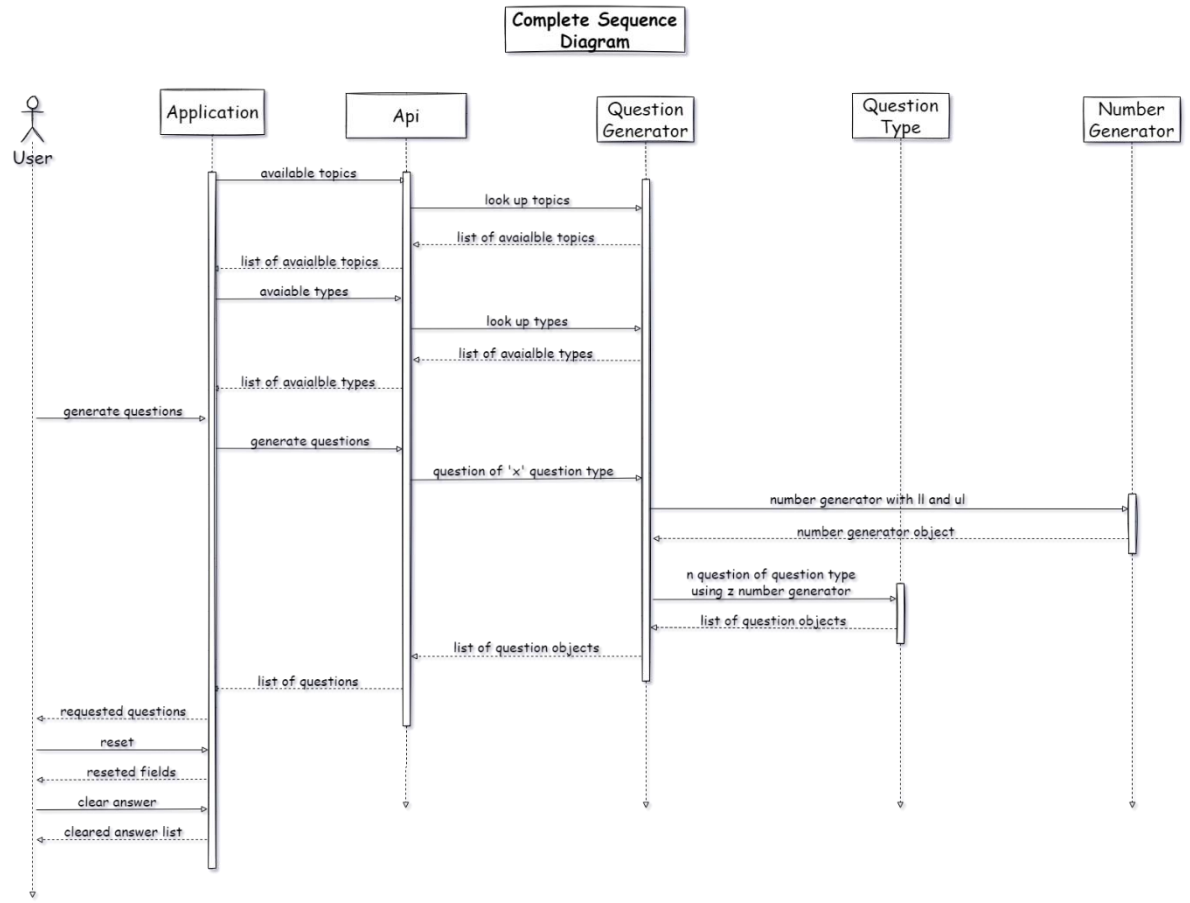




Sequence Diagram

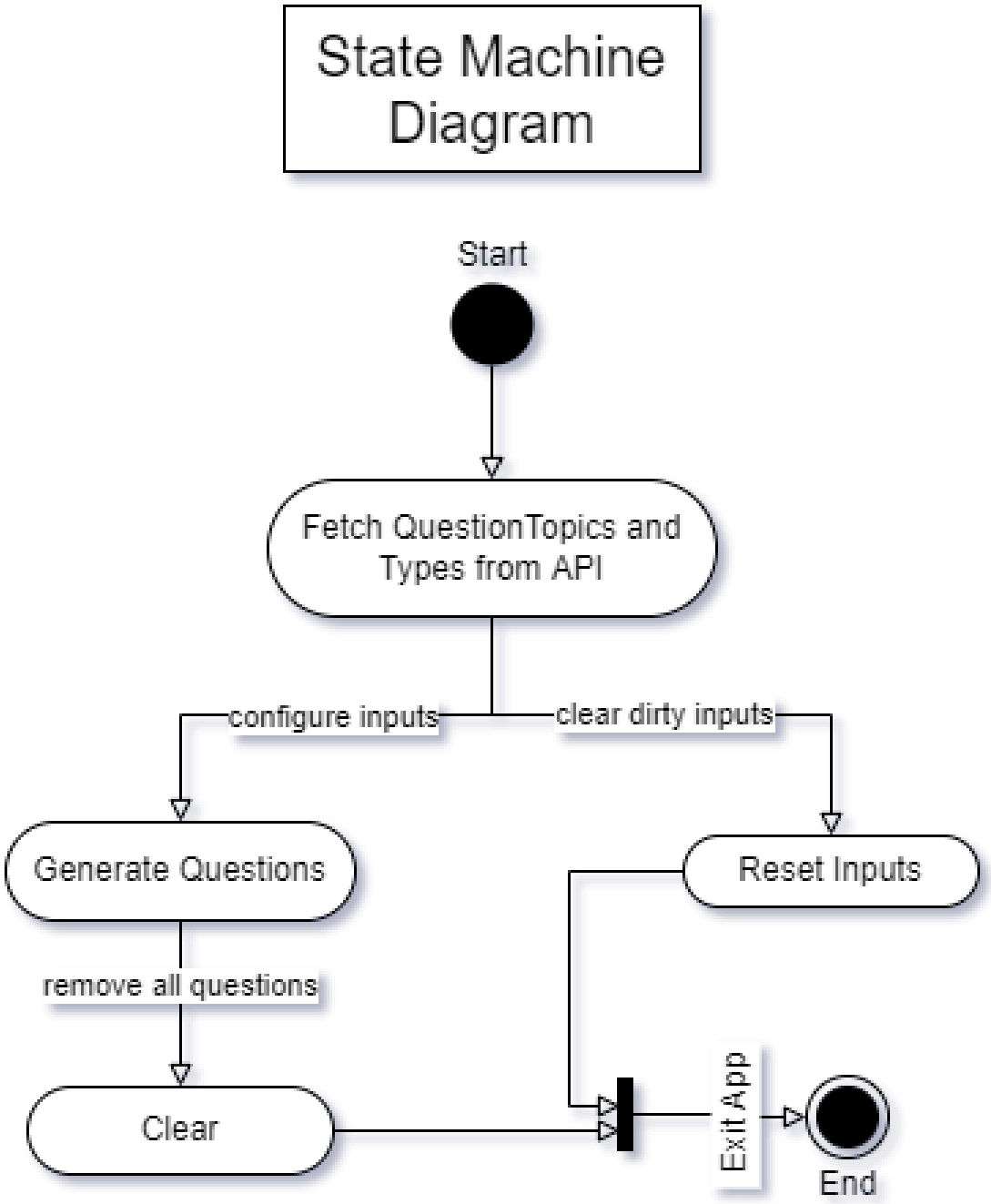
Sequence Diagram is a diagram that explains the order of interactions that the any actor is going to have with the system. The diagram shows how—and in what order—a group of objects works together.




Complete Sequence Diagram
User
Application
Api
Question Generator
Question Type
Number Generator
available topics
look up topics
list of avaialble topics
list of avaialble topics
avaiable types
look up types
list of avaialble types
list of avaialble types
generate questions
generate questions
question of 'x' question type
number generator with ll and ul
number generator object
n question of question type using z number generator
list of question objects
list of question objects
list of questions
requested questions
reset
reseted fields
clear answer
cleared answer list

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
################################
```

```python
@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.RangedIntegerNumberGen
erator(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.RangedNumberGe
nerator):
    """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_neg
ative=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

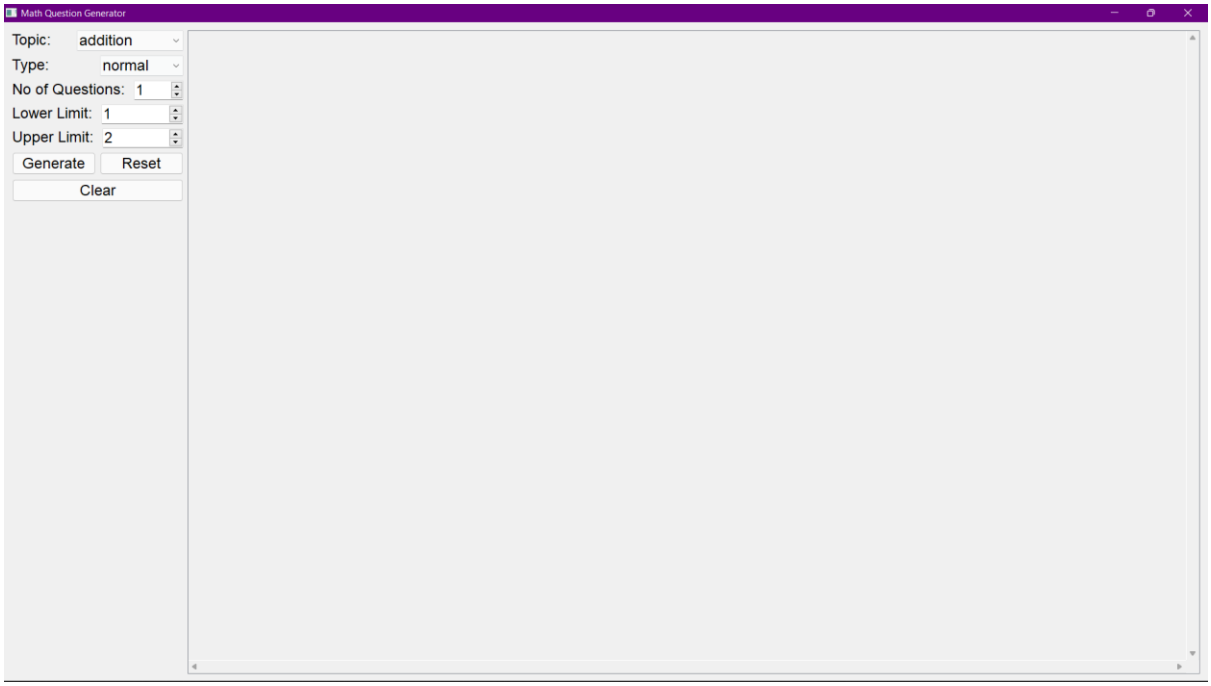
```
# question_strategies/subtraction.py
# In-built imports
import random
from typing import Type, Union

# 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"
```

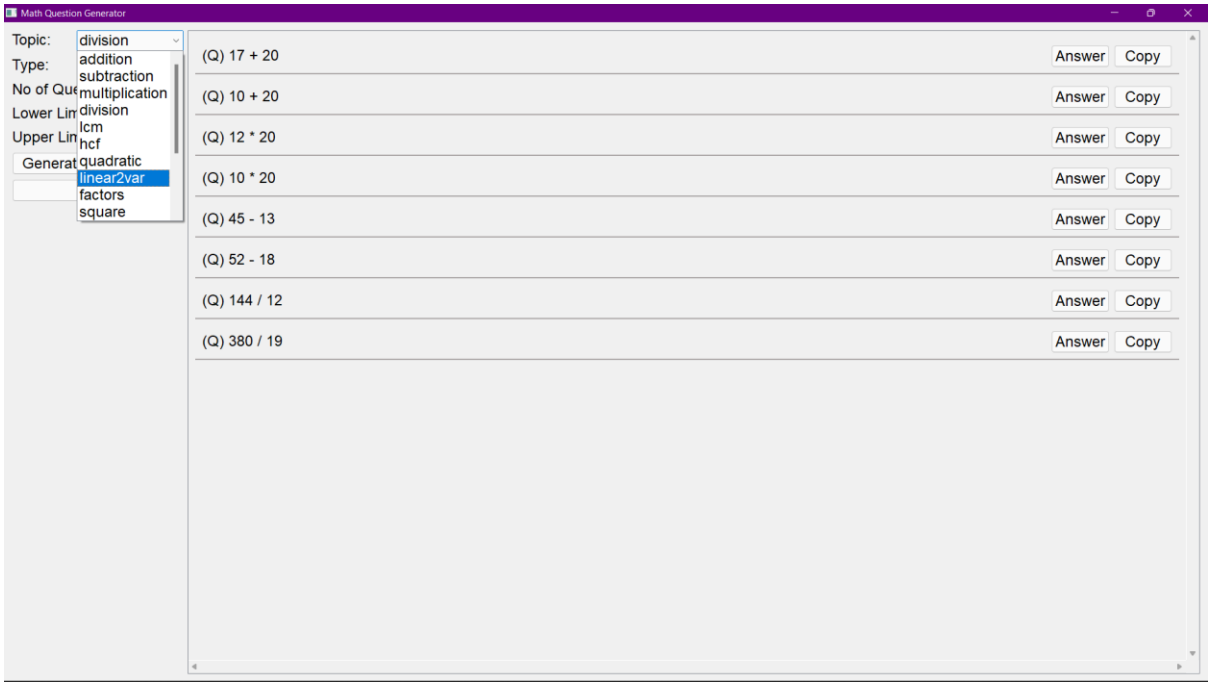
```
    }
    def __init__(self,
number_generator_cls:question.numGenType,
number_of_nums:int=2) -> None:
        super().__init__()
        self.number_generator_obj = number_generator_cls
        self.operator = "-"
        if number_of_nums < 2: raise
Exception("number_of_nums must be 2 or greater")
        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
}
```

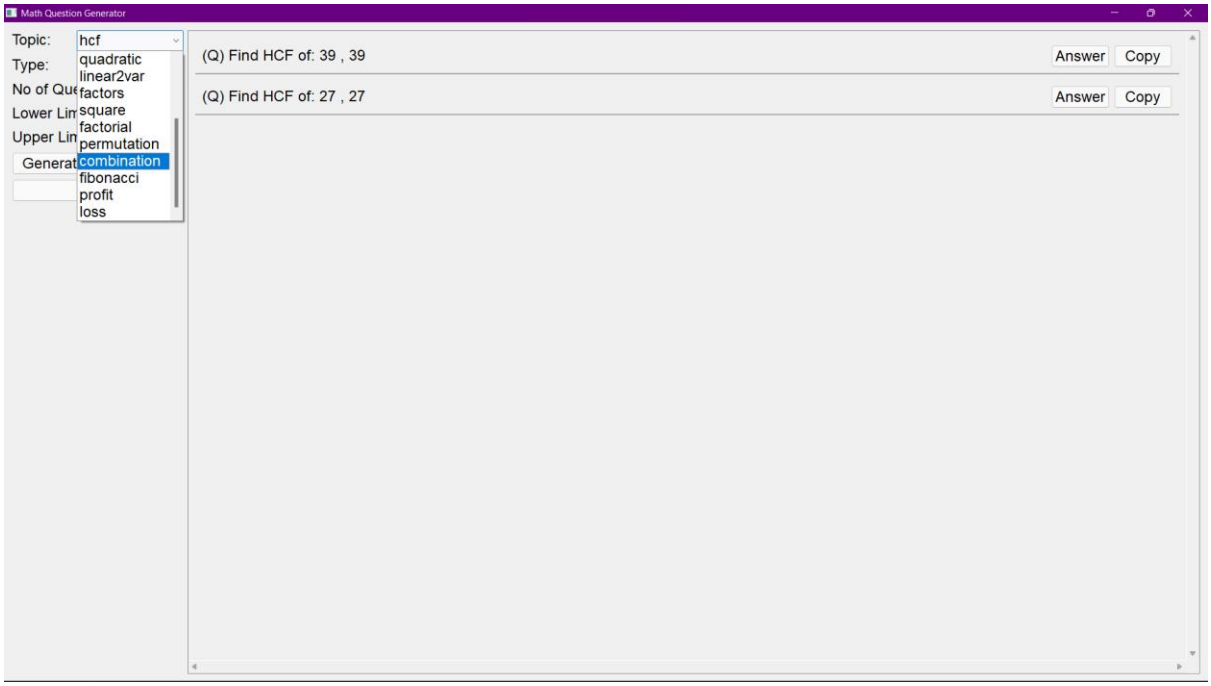
Screen Layout

- Start page

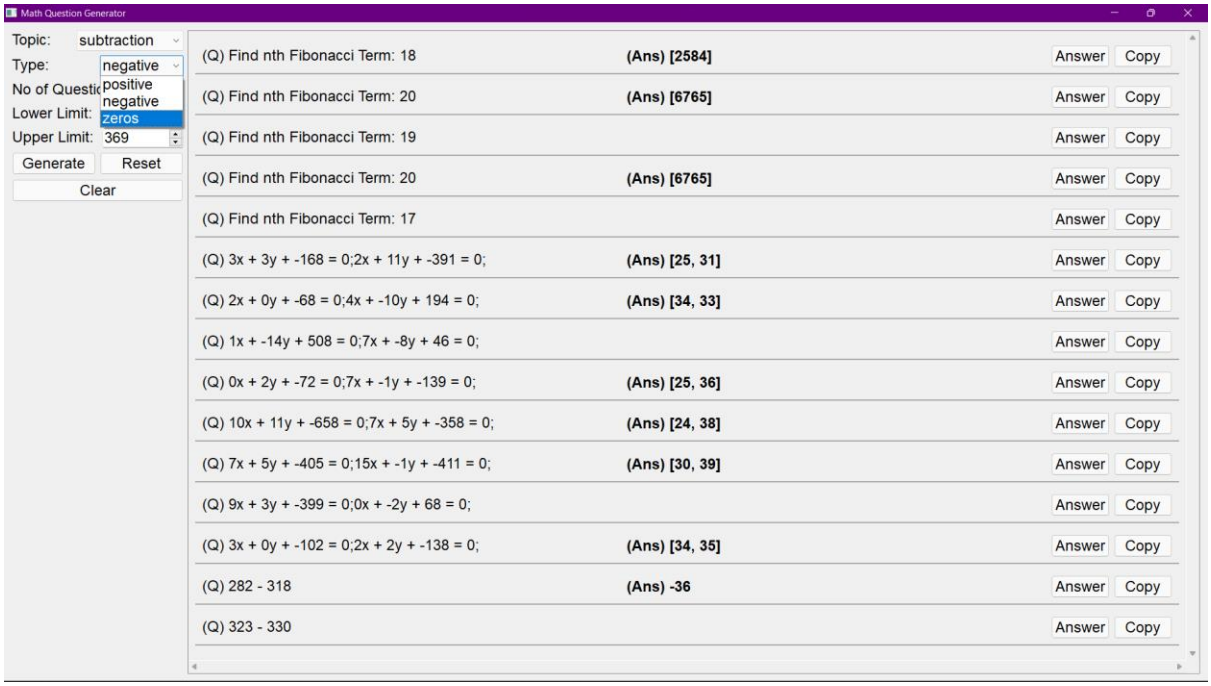


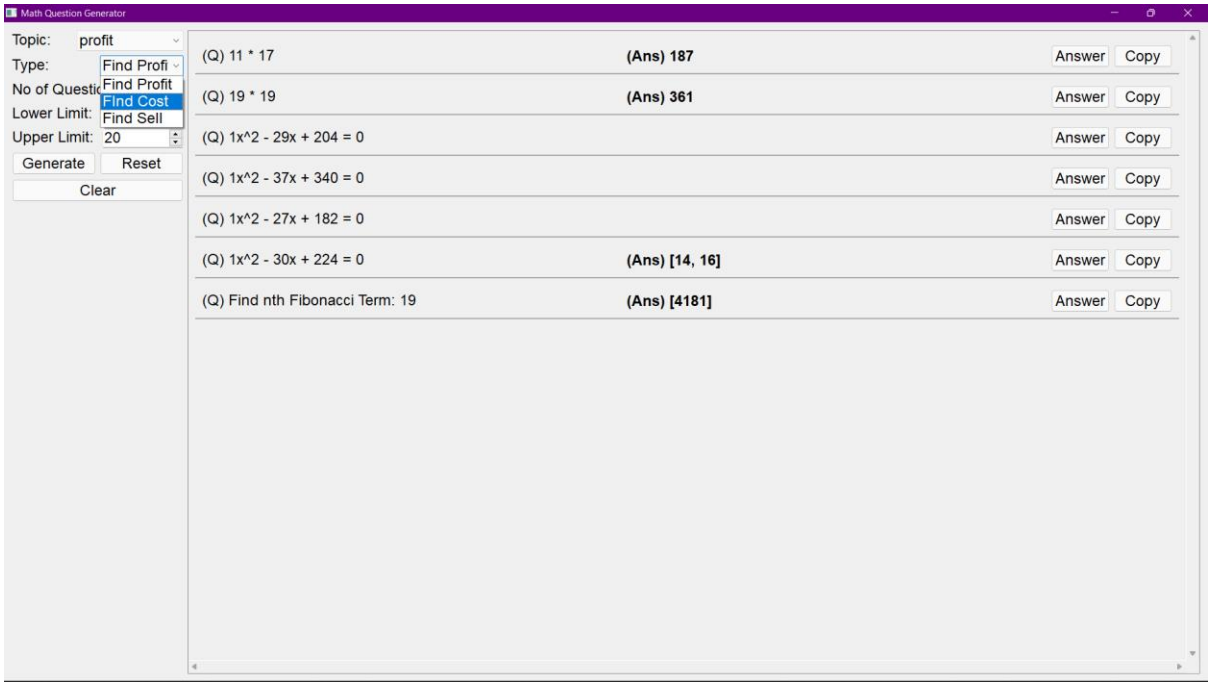
- Select Topic



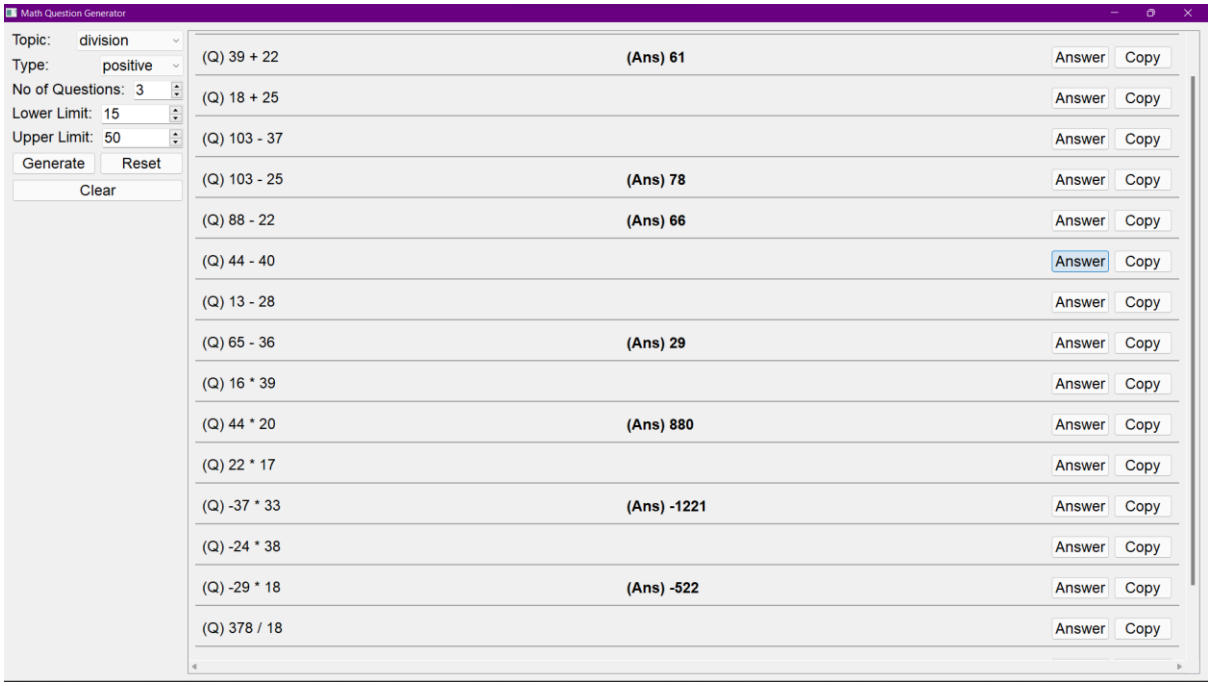


- Select Type

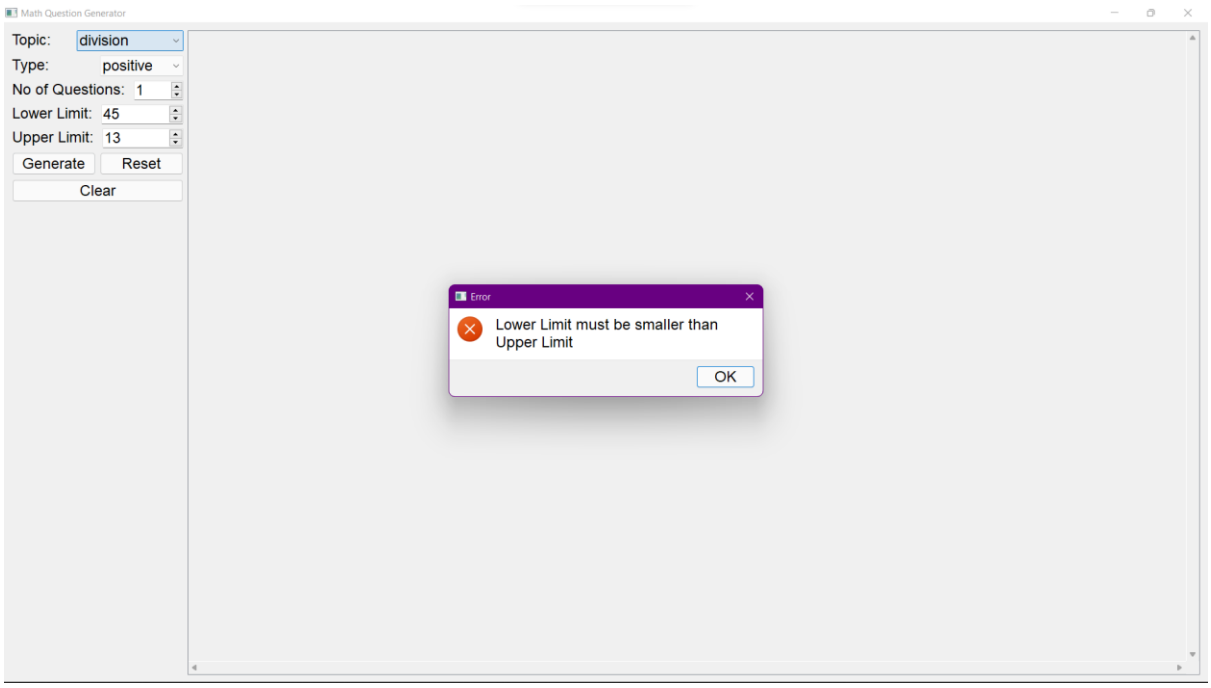

Math Question Generator
Topic: subtraction
Type: negative
positive
negative
zeros
No of Questic
Lower Limit:
Upper Limit: 369
Generate
Reset
Clear
(Q) Find nth Fibonacci Term: 18 (Ans) [2584] Answer Copy
(Q) Find nth Fibonacci Term: 20 (Ans) [6765] Answer Copy
(Q) Find nth Fibonacci Term: 19 Answer Copy
(Q) Find nth Fibonacci Term: 20 (Ans) [6765] Answer Copy
(Q) Find nth Fibonacci Term: 17 Answer Copy
(Q) 3x + 3y + -168 = 0;2x + 11y + -391 = 0; (Ans) [25, 31] Answer Copy
(Q) 2x + 0y + -68 = 0;4x + -10y + 194 = 0; (Ans) [34, 33] Answer Copy
(Q) 1x + -14y + 508 = 0;7x + -8y + 46 = 0; Answer Copy
(Q) 0x + 2y + -72 = 0;7x + -1y + -139 = 0; (Ans) [25, 36] Answer Copy
(Q) 10x + 11y + -658 = 0;7x + 5y + -358 = 0; (Ans) [24, 38] Answer Copy
(Q) 7x + 5y + -405 = 0;15x + -1y + -411 = 0; (Ans) [30, 39] Answer Copy
(Q) 9x + 3y + -399 = 0;0x + -2y + 68 = 0; Answer Copy
(Q) 3x + 0y + -102 = 0;2x + 2y + -138 = 0; (Ans) [34, 35] Answer Copy
(Q) 282 - 318 (Ans) -36 Answer Copy
(Q) 323 - 330 Answer Copy


Math Question Generator
Topic: profit
Type: Find Profi
Find Profit
Find Cost
Find Sell
No of Questic
Lower Limit:
Upper Limit: 20
Generate
Reset
Clear
(Q) 11 * 17 (Ans) 187 Answer Copy
(Q) 19 * 19 (Ans) 361 Answer Copy
(Q) 1x^2 - 29x + 204 = 0 Answer Copy
(Q) 1x^2 - 37x + 340 = 0 Answer Copy
(Q) 1x^2 - 27x + 182 = 0 Answer Copy
(Q) 1x^2 - 30x + 224 = 0 (Ans) [14, 16] Answer Copy
(Q) Find nth Fibonacci Term: 19 (Ans) [4181] Answer Copy

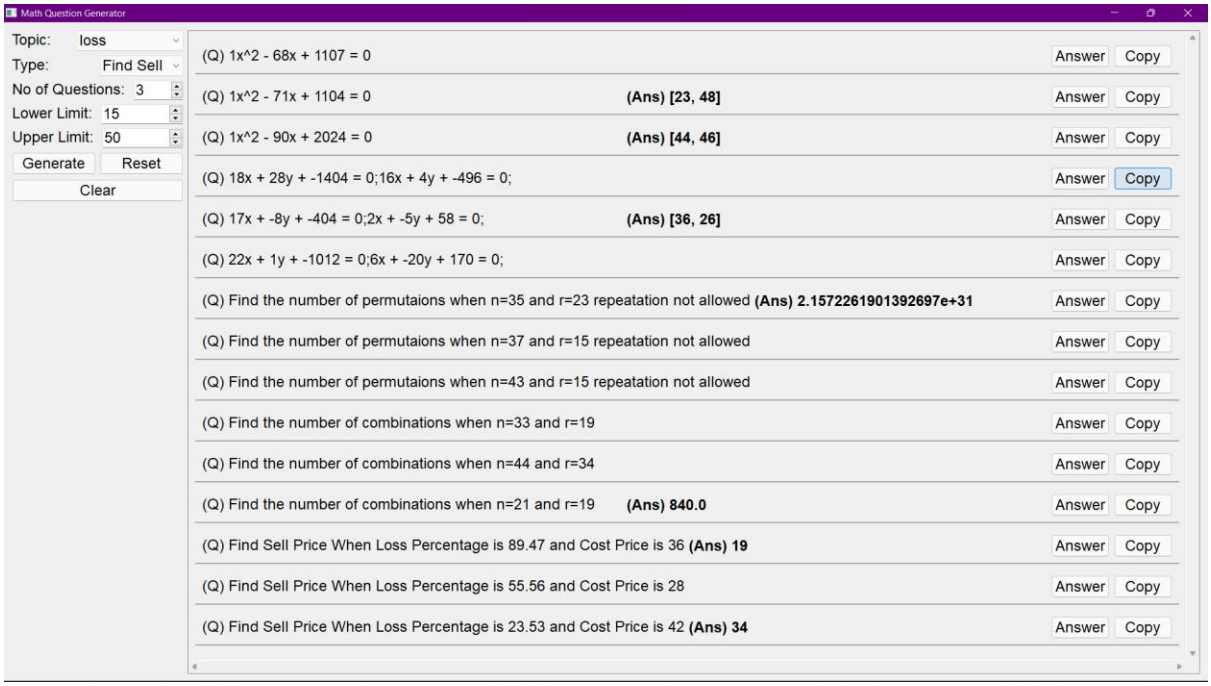
- Show Answer

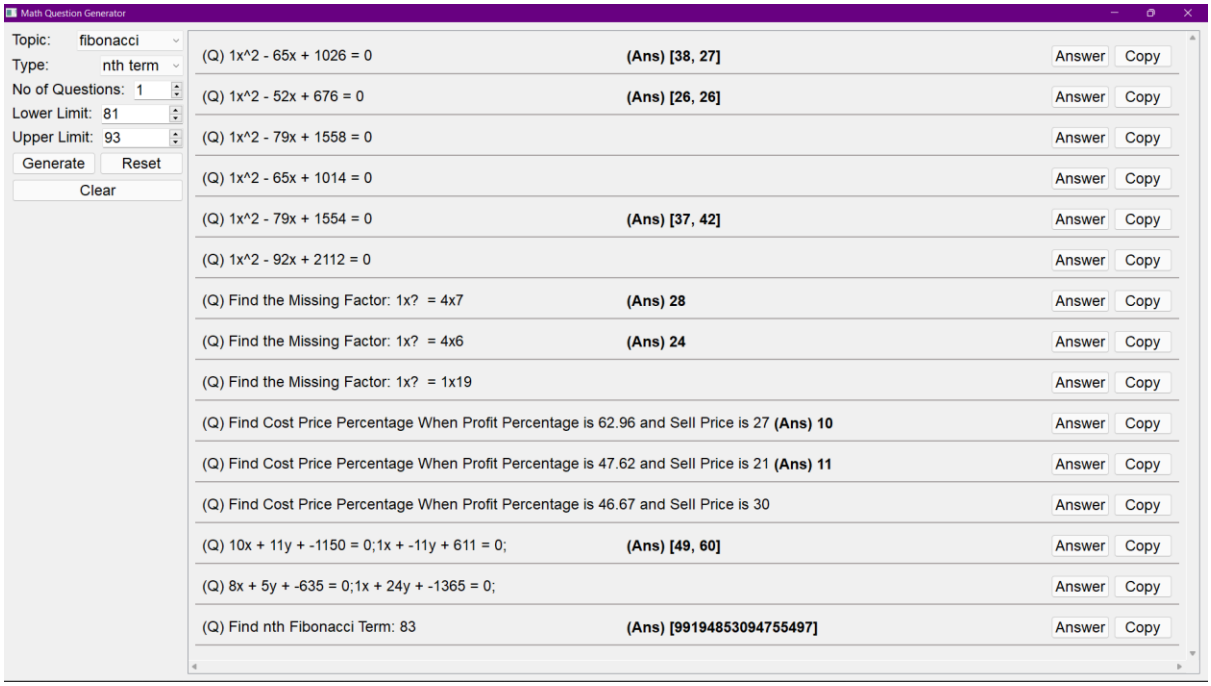


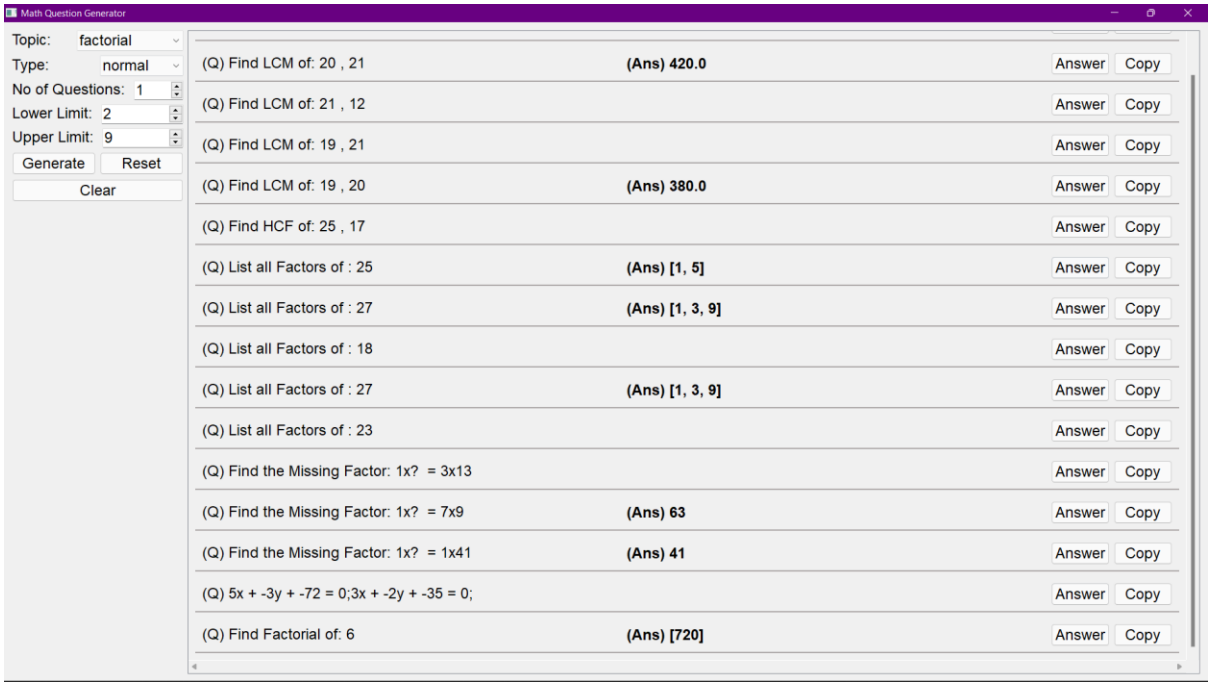
- Limit Error



- Usage Examples



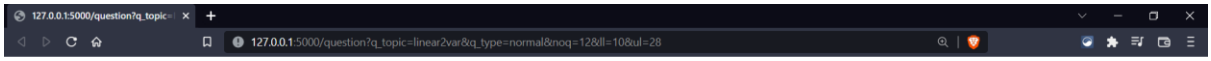



Math Question Generator
Topic: factorial
Type: normal
No of Questions: 1
Lower Limit: 2
Upper Limit: 9
Generate
Reset
Clear
(Q) Find LCM of: 20 , 21 (Ans) 420.0 Answer Copy
(Q) Find LCM of: 21 , 12 Answer Copy
(Q) Find LCM of: 19 , 21 Answer Copy
(Q) Find LCM of: 19 , 20 (Ans) 380.0 Answer Copy
(Q) Find HCF of: 25 , 17 Answer Copy
(Q) List all Factors of : 25 (Ans) [1, 5] Answer Copy
(Q) List all Factors of : 27 (Ans) [1, 3, 9] Answer Copy
(Q) List all Factors of : 18 Answer Copy
(Q) List all Factors of : 27 (Ans) [1, 3, 9] Answer Copy
(Q) List all Factors of : 23 Answer Copy
(Q) Find the Missing Factor: 1x? = 3x13 Answer Copy
(Q) Find the Missing Factor: 1x? = 7x9 (Ans) 63 Answer Copy
(Q) Find the Missing Factor: 1x? = 1x41 (Ans) 41 Answer Copy
(Q) 5x + -3y + -72 = 0;3x + -2y + -35 = 0; Answer Copy
(Q) Find Factorial of: 6 (Ans) [720] Answer Copy

- Browser Example

[{"answer":[1,2,3,4,6,8,12],"question_string":"List all Factors of : 24","type":"Factors_Type2"},
{"answer":[1,2,3,4,6,8,12],"question_string":"List all Factors of : 24","type":"Factors_Type2"},
{"answer":[1,2,3,4,6,8,12],"question_string":"List all Factors of : 24","type":"Factors_Type2"},
{"answer":[1,2,3,4,6,8,12],"question_string":"List all Factors of : 24","type":"Factors_Type2"},
{"answer":[1,2,3,4,6,8,12],"question_string":"List all Factors of : 24","type":"Factors_Type2"},
{"answer":[1,2,3,4,6,8,12],"question_string":"List all Factors of : 24","type":"Factors_Type2"},
{"answer":[1,2,3,4,6,8,12],"question_string":"List all Factors of : 24","type":"Factors_Type2"},
{"answer":[1,2,3,4,6,8,12],"question_string":"List all Factors of : 24","type":"Factors_Type2"},
{"answer":[1,2,3,4,6,8,12],"question_string":"List all Factors of : 24","type":"Factors_Type2"},
{"answer":[1,2,3,4,6,8,12],"question_string":"List all Factors of : 24","type":"Factors_Type2"}]

[{"answer":48841,"question_string":"Find Square of: 221","type":"SquareType1"},
{"answer":19321,"question_string":"Find Square of: 139","type":"SquareType1"},
{"answer":26896,"question_string":"Find Square of: 164","type":"SquareType1"}]



[{"answer":[13,12],"question_string":"4x + -12y + 92 = 0;1x + 0y + -13 = 0;","type":"Linear_2_Var_Type1"},{"answer":
[21,10],"question_string":"5x + 4y + -145 = 0;13x + 0y + -273 = 0;","type":"Linear_2_Var_Type1"},{"answer":
[16,19],"question_string":"5x + 4y + -156 = 0;7x + 1y + -131 = 0;","type":"Linear_2_Var_Type1"},{"answer":
[14,21],"question_string":"8x + 12y + -364 = 0;3x + 0y + -42 = 0;","type":"Linear_2_Var_Type1"},{"answer":
[25,28],"question_string":"3x + -10y + 205 = 0;0x + 2y + -56 = 0;","type":"Linear_2_Var_Type1"},{"answer":
[12,25],"question_string":"13x + 15y + -531 = 0;1x + 4y + -112 = 0;","type":"Linear_2_Var_Type1"},{"answer":
[21,16],"question_string":"1x + 6y + -117 = 0;3x + -8y + 65 = 0;","type":"Linear_2_Var_Type1"},{"answer":
[18,12],"question_string":"15x + -4y + -222 = 0;2x + 7y + -120 = 0;","type":"Linear_2_Var_Type1"},{"answer":
[25,20],"question_string":"8x + 9y + -380 = 0;2x + 2y + -90 = 0;","type":"Linear_2_Var_Type1"},{"answer":
[20,17],"question_string":"8x + 0y + -160 = 0;7x + -8y + -4 = 0;","type":"Linear_2_Var_Type1"},{"answer":
[20,22],"question_string":"3x + -6y + 72 = 0;8x + 8y + -336 = 0;","type":"Linear_2_Var_Type1"},{"answer":
[10,24],"question_string":"1x + 0y + -10 = 0;4x + 17y + -448 = 0;","type":"Linear_2_Var_Type1"}]

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

Lukew3 – mathgenerator github repository

Januschung – math-worksheet-generator github repository

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

Draw.io – https://app.diagrams.net/