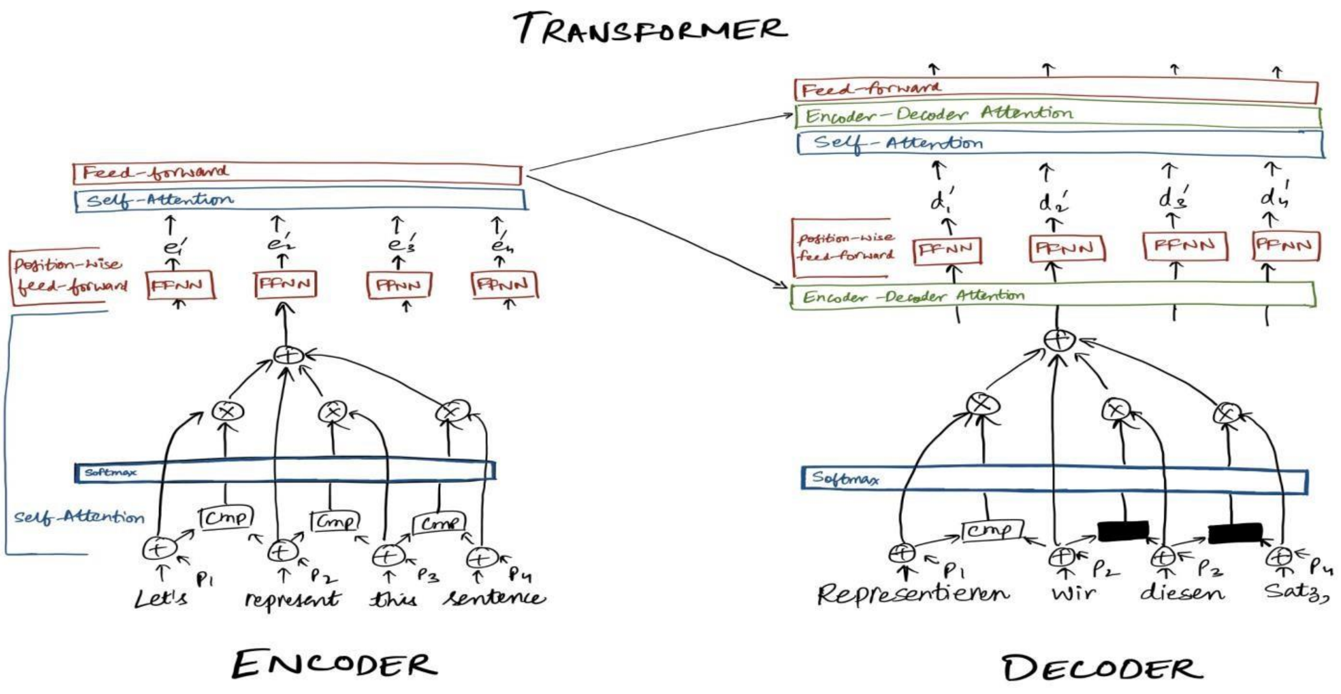
Transformer-based End-to-End Question Generation -

:ABSTRACT-:

Question Generation (QG) is an important task in Natural Language Processing (NLP) that involves generating questions automatically when given a context paragraph. While many techniques exist for the task of QG, they employ complex model architectures, extensive features, and additional mechanisms to boost model performance. In this work, we show that transformer-based finetuning techniques can be used to create robust question generation systems using only a single pretrained language model, without the use of additional mechanisms, answer metadata, and extensive features. Our best model outperforms previous more complex RNN-based Seq2Seq models, with an 8.62 and a 14.27 increase in METEOR and ROUGE L scores, respectively. We show that it also performs on par with Seq2Seq models that employ answer awareness and other special mechanisms, despite being only a single-model system. We analyze how various factors affect the model's performance, such as input data formatting, the length of the context paragraphs, and the use of answer-awareness. Lastly, we also look into the model's failure modes and identify possible reasons why the model fails.



Introduction:-

- ► Question Generation involves generating questions automatically when given a context paragraph.
- ► The ability to ask meaningful questions provides evidence towards comprehension within an Artificial Intelligence (AI) model.
- ► One use case is the implementation of assisted / guided learning.
- ► Another use case is online assistants, such as travel assistants.
- ► Question generation is an important natural language processing task. ► To generate a question generation model most widely-used techniques are Deep Learning-based approaches involving Sequence-to-Sequence (Seq2Seq).
- ► While all of these techniques are robust, they all employ complex models, extra features, and additional mechanisms that make them harder to train and expensive to reproduce.
- ► We make use of transformers for model implementation.

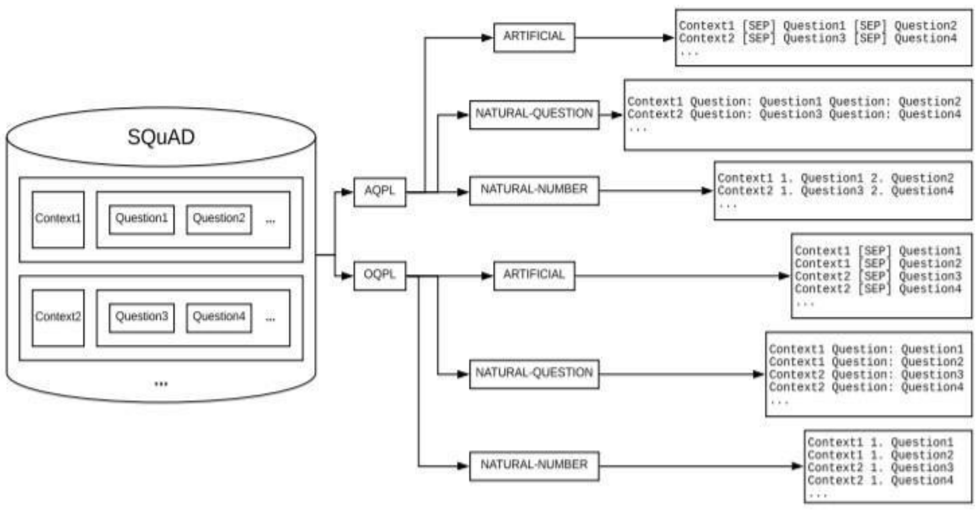
Model Training:-

- ► Dataset used= Stanford Question Answering Dataset (SQuAD). We format SQuAD such that it appears similar to input data for language modeling.
- ► It contains context paragraphs each with a set of questions and answers.
- ► It contains more than 100,000 questions.
- ► For our model The entire dataset is transformed into a continuous body of text.
- ► A Question and its context are separated by a delimiter.

► During training, this delimiter allows the model to properly distinguish between the context and question, while during prediction, it can be used as

a marker at the end of some input text to invoke question generation behaviour in the model

► Here is our data preparation pipeline for SQuAD



SQuAD
Context1
Question1
Question2
...
Context2
Question3
Question4
...
AQPL
OQPL
ARTIFICIAL
NATURAL-QUESTION
NATURAL-NUMBER
ARTIFICIAL
NATURAL-QUESTION
NATURAL-NUMBER
Context1 [SEP] Question1 [SEP] Question2
Context2 [SEP] Question3 [SEP] Question4
...
Context1 Question: Question1 Question: Question2
Context2 Question: Question3 Question: Question4
...
Context1 1. Question1 2. Question2
Context2 1. Question3 2. Question4
...
Context1 [SEP] Question1
Context1 [SEP] Question2
Context2 [SEP] Question3
Context2 [SEP] Question4
...
Context1 Question: Question1
Context1 Question: Question2
Context2 Question: Question3
Context2 Question: Question4
...
Context1 1. Question1
Context1 1. Question2
Context2 1. Question3
Context2 1. Question4
...

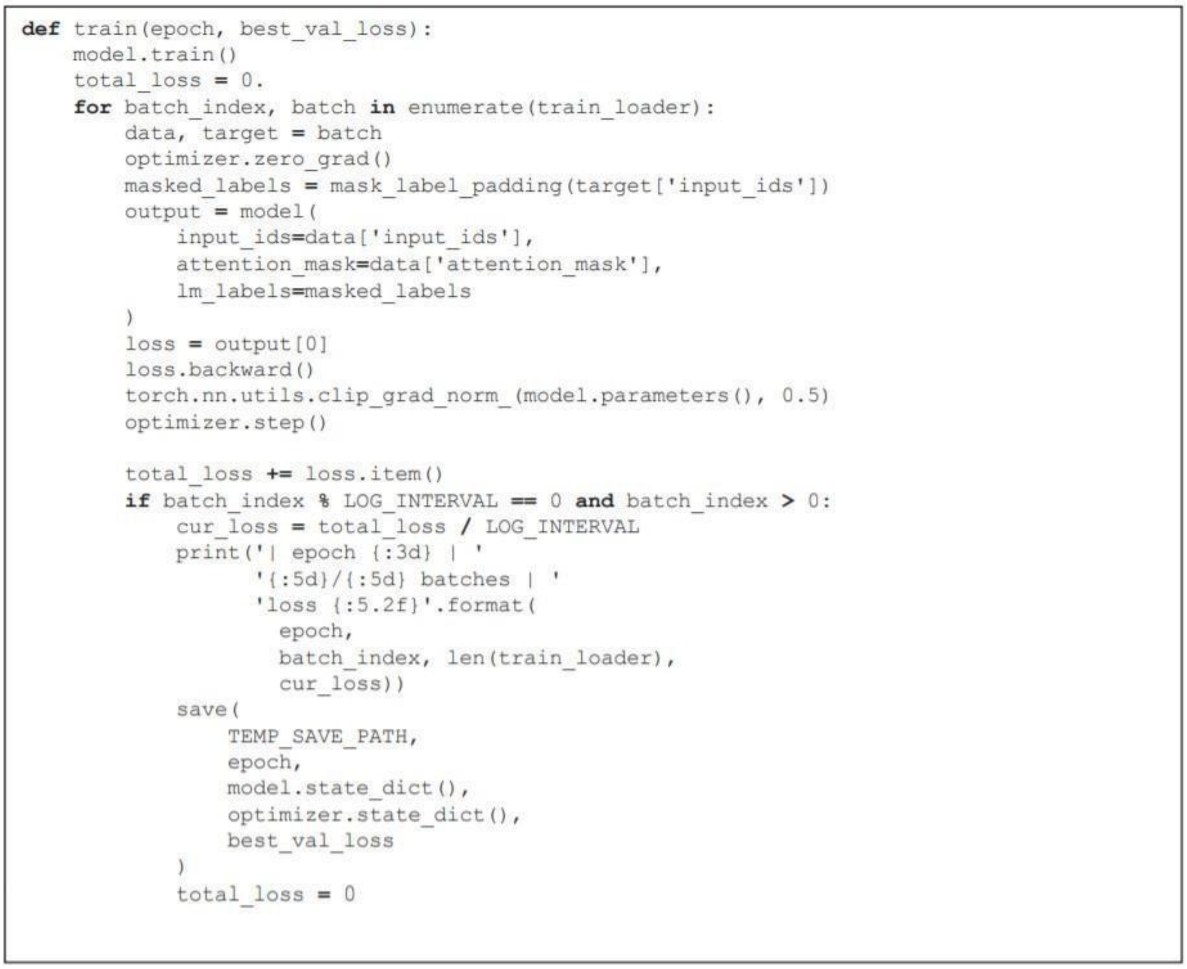
Questions Per Line:-There can be several possible questions associated with a single paragraph. We experiment with two ways to flatten this many-to-one relationship in the formatted data:

All Questions Per Line (AQPL) :-A single training example consists of a context paragraph with all of its associated questions placed immediately after it, separated from one another with the selected delimiter. While this avoids duplication of context and thus results in faster training time, it may potentially result in the model no longer being able to attend to earlier tokens as its context window moves further away from the beginning of the input paragraph.

One Question Per Line (OQPL):-Each context paragraph is duplicated for each of its associated questions, such that for a single training example, there is only one context and one question. For many cases, this may alleviate the moving context window problem raised with AQPL, as the length of a single training example is reduced to the length of an input paragraph plus the length of a single associated question. However, this format does result in a longer training time due to the duplicated contexts increasing the size of the final formatted dataset.

Model Setup and Finetuning:-

- Our pre-trained model='t5-base'. We fine tune this model for end to end question generation.
- "Fine-tuning" in NLP refers to the procedure of re-training a pre-trained language model using your own custom data.
- We trained our model for 20 epochs with initial learning rate=1e-3.
- Learning rate can vary while training based on convergence rate.
- Here is our small part of the code for training the model. (Similar to that we used in Assignment 3).



```python
def train(epoch, best_val_loss):
    model.train()
    total_loss = 0.
    for batch_index, batch in enumerate(train_loader):
        data, target = batch
        optimizer.zero_grad()
        masked_labels = mask_label_padding(target['input_ids'])
        output = model(
            input_ids=data['input_ids'],
            attention_mask=data['attention_mask'],
            lm_labels=masked_labels
        )
        loss = output[0]
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), 0.5)
        optimizer.step()

        total_loss += loss.item()
        if batch_index % LOG_INTERVAL == 0 and batch_index > 0:
            cur_loss = total_loss / LOG_INTERVAL
            print('| epoch {:3d} | '
                  '{:5d}/{:5d} batches | '
                  'loss {:5.2f}'.format(
                    epoch,
                    batch_index, len(train_loader),
                    cur_loss))
            save(
                TEMP_SAVE_PATH,
                epoch,
                model.state_dict(),
                optimizer.state_dict(),
                best_val_loss
            )
            total_loss = 0
```

Model Generation, Metrics & Evaluation:-

► We will set the model temperature to be 0.6 as higher value will result in more randomness and lower value result in greedy behaviour.

► We use the top-p nucleus sampling method with a value of p = 0.9. ► Each generation loop is terminated either when the model generates the newline character '\n', or when the model reaches a generation length of 32 tokens.

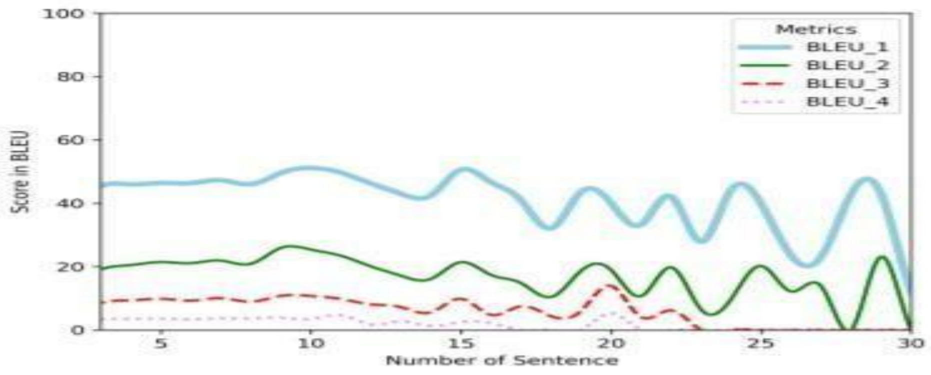
► We manually set this maximum length in order to terminate generation sessions that are stuck in token span loops and do not reach the '\n' endoftext token on their own.

Results and discussions:-

- The best performing model is the One Question Per Line (OQPL) model with number delimiters, achieving the highest score for BLEU 2, BLEU 3, BLEU 4 and METEOR.
- Even the BLEU score for All question per line(AQPL) is not statistically much behind.
- The choice between data formatting (OQPL vs AQPL) only matters marginally, given that the context length does not approach the maximum sequence length of the model.

Optimal Context Length

- A short context paragraph will have a more apparent subject, which can be directly used by the model's context-copy mechanism in order to generate good questions.
- If the model encounters a long context paragraph where the context is not clear, the context-copy mechanism that the model usually employs will have a hard time pinpointing exact attention positions from where it bases its generated questions from and it will result in low BLEU score against the expected high scores.



BLEU Scores for each length

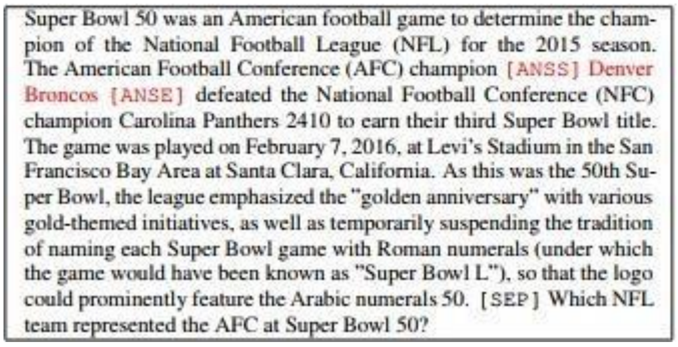
We can see the length of a sentence around 15 is optimal.

Answer Awareness:-

► Answer-awareness refers to the usage of the answer's position or the answer to the question itself, alongside the context paragraph, as input to the model for question generation.

► Given that a number of well-performing previous studies on question generation used answer awareness, we shall also test whether the singletransformer method employed here would benefit from this additional feature.

► An OQPL artificial based formatting scheme is employed, where the start position of the answer within the context is marked with a special answerstart token ([ANSS]), while the end position of the answer within the context is marked with another special answer-end token ([ANSE]). A sample input context paragraph is shown in the figure below, with the answer-awareness tokens.



Super Bowl 50 was an American football game to determine the champion of the National Football League (NFL) for the 2015 season. The American Football Conference (AFC) champion [ANSS] Denver Broncos [ANSE] defeated the National Football Conference (NFC) champion Carolina Panthers 2410 to earn their third Super Bowl title. The game was played on February 7, 2016, at Levi's Stadium in the San Francisco Bay Area at Santa Clara, California. As this was the 50th Super Bowl, the league emphasized the "golden anniversary" with various gold-themed initiatives, as well as temporarily suspending the tradition of naming each Super Bowl game with Roman numerals (under which the game would have been known as "Super Bowl L"), so that the logo could prominently feature the Arabic numerals 50. [SEP] Which NFL team represented the AFC at Super Bowl 50?

Fig: A sample training example for answer-aware question generation training. The marked answer span is highlighted in red. The above uses the ARTIFICIAL delimiter and the OQPL format. The above text has been adopted from SQuAD dataset (Rajpurkar et al., 2016). ► Then, the same fine-tuning setup as the original OQPL artificial model was followed, evaluating on the BLEU and ROUGE_L scores. A summary of the fine-tuning results are provided in the table below:

Model	BLEU_1	BLEU_2	BLEU_3	BLEU_4	ROUGE_L
OQPL Standard	55.60	31.03	16.56	7.89	44.41
OQPL Answer-Aware	36.07	18.83	10.95	6.40	39.80

- We can clearly see that the answer-awareness models perform significantly worse considering the BLEU scores, while they perform marginally worse (than the original OQPL model) if we take the ROUGE_L.
- Without concrete foundation, we suppose that this is because the model has no inherent idea what to do with the answer-awareness information, and unlike true answer-aware models like UniLM (Dong et al., 2019), no explicit mechanism that puts importance to the answer-awareness is present in the model. The model (here) eventually, in the end, still performs better without answer-awareness.

Conclusions:-

- A simple single Transformer-based question generation model is able to outperform more complex Seq2Seq methods without the need for additional features, techniques, and training steps.
- It would be better to train models using the three larger GPT-2 model sizes.
- It would be better to train the model also on datasets other than SQuAD that have larger context lengths.
- It is recommended that an explicit mechanism for handling answer-aware features be added to the model using different ways like embedding the

features separately and concatenating them with the produced hidden states of the transformer, etc.

► Lastly, it would be good to explore a possible use for this question generation model in natural language understanding tasks, for example, a pre-trained base model; since the capability of the model to ask questions about a text might imply that the model really has “some understanding” about the text.

:-OUR-TRAINING-PROCEDURE

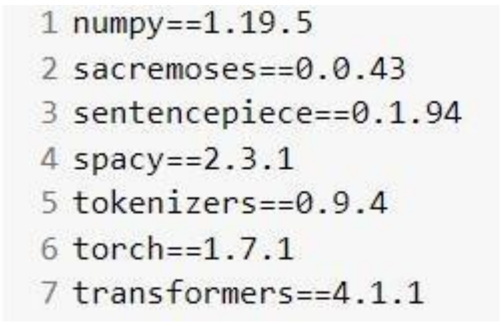
We used the same method to train model which is given in our paper.We used first a pre-trained basic model name (T5-BASE).Then we fine tuned it from scratch with a new merged dataset.

DATASET:-We used merged dataset of (SQuAD, RACE, CoQA, and MSMARCO)in csv format .Then we reduce dataset size because due to gpu time limit constraint in google collab we can not train complete model in full dataset.we made two dataset one for training and one for validation.

PLATFORM- We train our model in google collab with NVIDIA-TESLA-P100 gpu for 7 epochs.

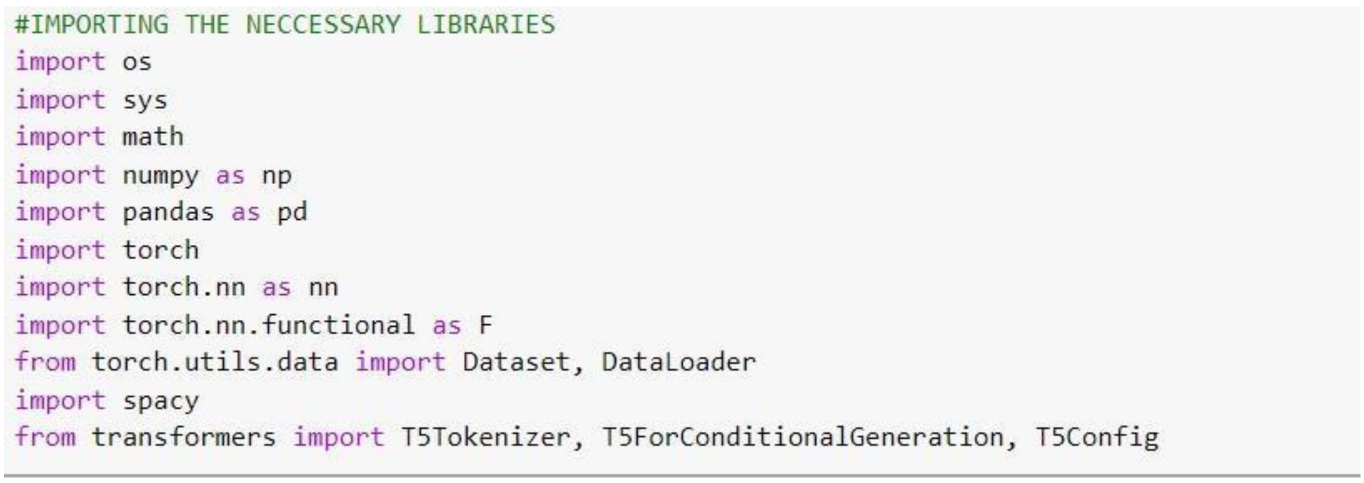
:-CODE-EXPLANATION:-

1.Library installation:-



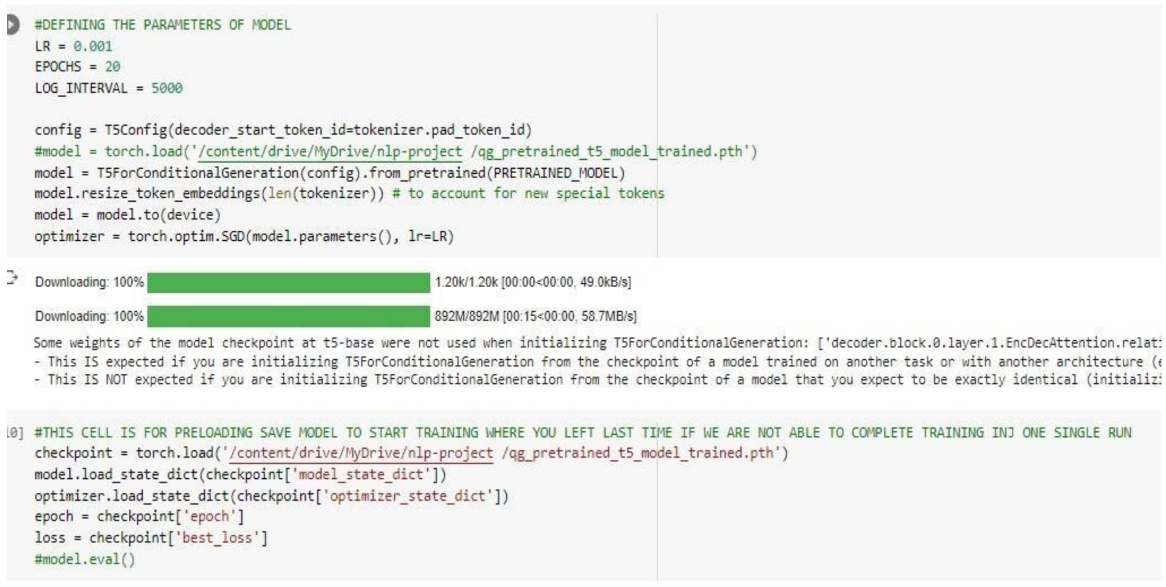
```
numpy==1.19.5
sacremoses==0.0.43
sentencepiece==0.1.94
spacy==2.3.1
tokenizers==0.9.4
torch==1.7.1
transformers==4.1.1
```

2.Importing libraries:-



```python
#IMPORTING THE NECCESSARY LIBRARIES
import os
import sys
import math
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader
import spacy
from transformers import T5Tokenizer, T5ForConditionalGeneration, T5Config
```

MODEL-LOADING:-

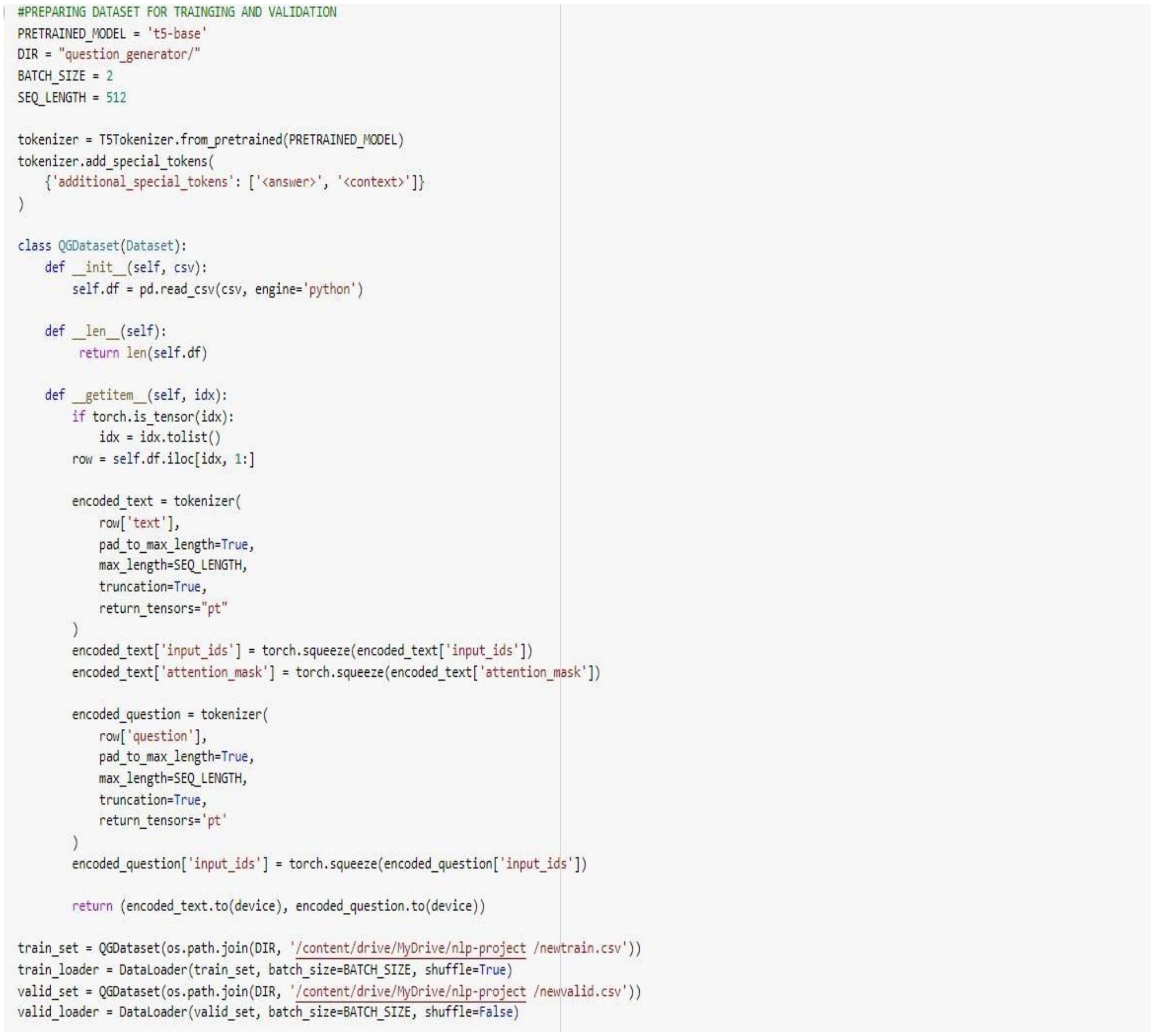


```python
#DEFINING THE PARAMETERS OF MODEL
LR = 0.001
EPOCHS = 20
LOG_INTERVAL = 5000

config = T5Config(decoder_start_token_id=tokenizer.pad_token_id)
#model = torch.load('/content/drive/MyDrive/nlp-project /qg_pretrained_t5_model_trained.pth')
model = T5ForConditionalGeneration(config).from_pretrained(PRETRAINED_MODEL)
model.resize_token_embeddings(len(tokenizer)) # to account for new special tokens
model = model.to(device)
optimizer = torch.optim.SGD(model.parameters(), lr=LR)
```

```
Downloading: 100% 1.20k/1.20k [00:00<00:00, 49.0kB/s]

Downloading: 100% 892M/892M [00:15<00:00, 58.7MB/s]
Some weights of the model checkpoint at t5-base were not used when initializing T5ForConditionalGeneration: ['decoder.block.0.layer.1.EncDecAttention.relati
- This IS expected if you are initializing T5ForConditionalGeneration from the checkpoint of a model trained on another task or with another architecture (e
- This IS NOT expected if you are initializing T5ForConditionalGeneration from the checkpoint of a model that you expect to be exactly identical (initializi
```

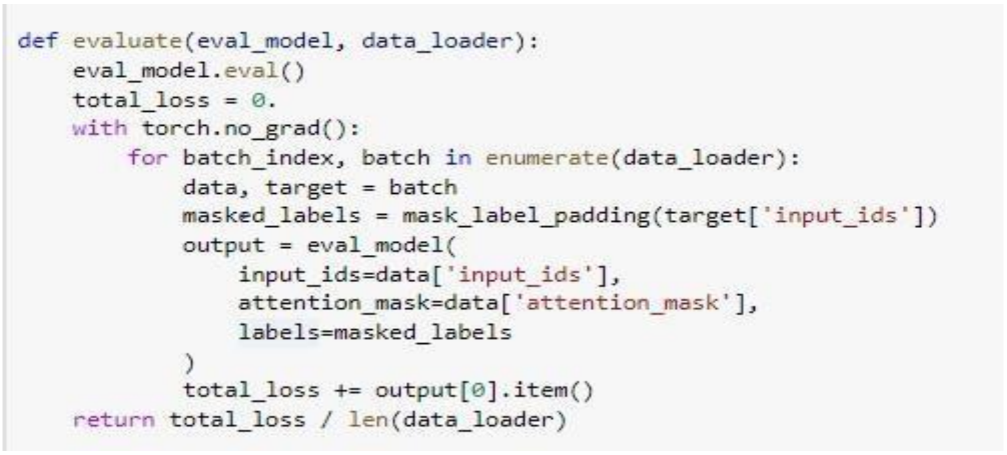
```python
10] #THIS CELL IS FOR PRELOADING SAVE MODEL TO START TRAINING WHERE YOU LEFT LAST TIME IF WE ARE NOT ABLE TO COMPLETE TRAINING INJ ONE SINGLE RUN
checkpoint = torch.load('/content/drive/MyDrive/nlp-project /qg_pretrained_t5_model_trained.pth')
model.load_state_dict(checkpoint['model_state_dict'])
optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
epoch = checkpoint['epoch']
loss = checkpoint['best_loss']
#model.eval()
```

3.DATASET-PREPARATION:-



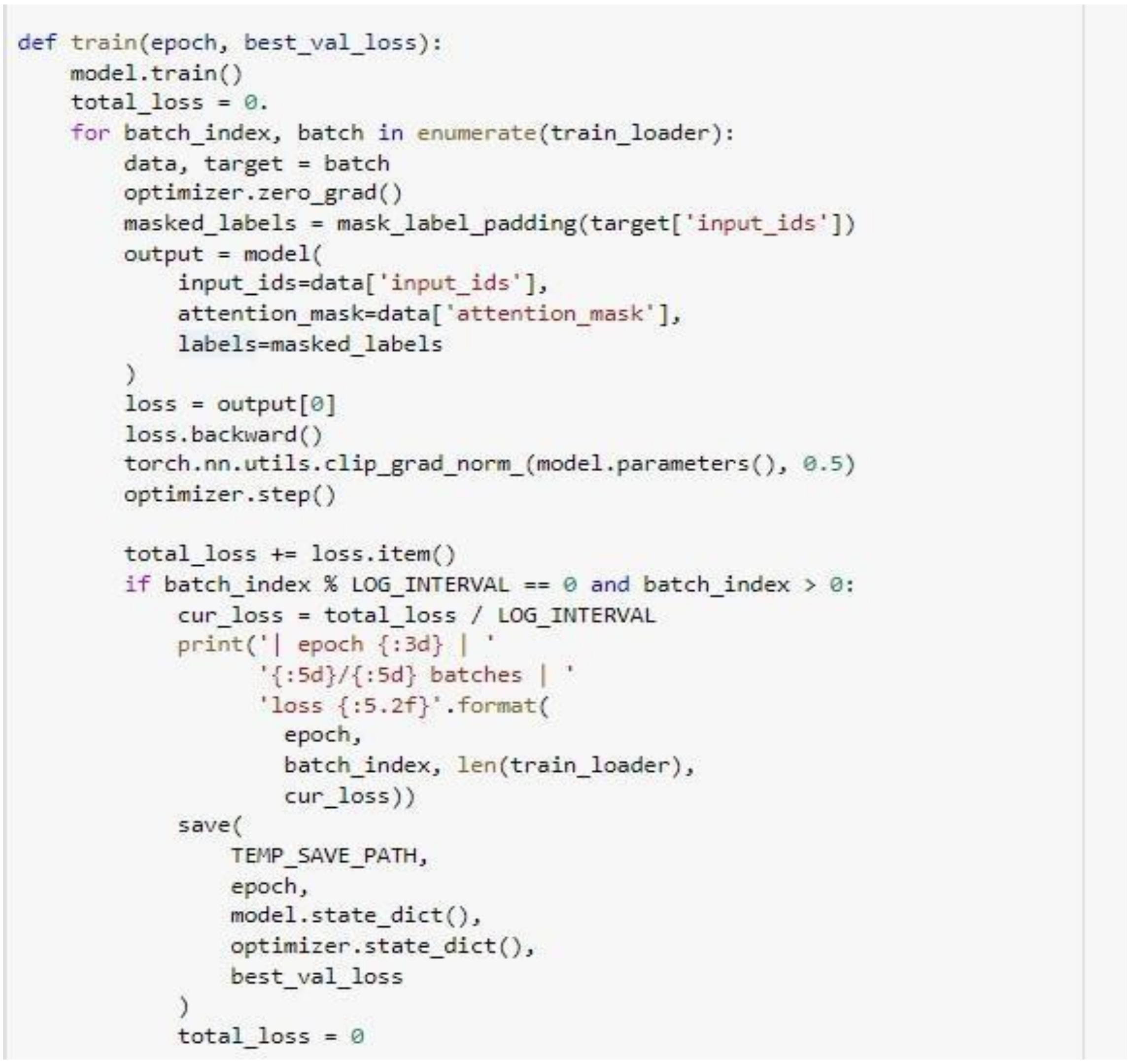
```
#PREPARING DATASET FOR TRAINGING AND VALIDATION
PRETRAINED_MODEL = 't5-base'
DIR = "question_generator/"
BATCH_SIZE = 2
SEQ_LENGTH = 512

tokenizer = T5Tokenizer.from_pretrained(PRETRAINED_MODEL)
tokenizer.add_special_tokens(
    {'additional_special_tokens': ['<answer>', '<context>']}
)

class QGDataset(Dataset):
    def __init__(self, csv):
        self.df = pd.read_csv(csv, engine='python')

    def __len__(self):
         return len(self.df)

    def __getitem__(self, idx):
        if torch.is_tensor(idx):
            idx = idx.tolist()
        row = self.df.iloc[idx, 1:]

        encoded_text = tokenizer(
            row['text'],
            pad_to_max_length=True,
            max_length=SEQ_LENGTH,
            truncation=True,
            return_tensors="pt"
        )
        encoded_text['input_ids'] = torch.squeeze(encoded_text['input_ids'])
        encoded_text['attention_mask'] = torch.squeeze(encoded_text['attention_mask'])

        encoded_question = tokenizer(
            row['question'],
            pad_to_max_length=True,
            max_length=SEQ_LENGTH,
            truncation=True,
            return_tensors='pt'
        )
        encoded_question['input_ids'] = torch.squeeze(encoded_question['input_ids'])

        return (encoded_text.to(device), encoded_question.to(device))

train_set = QGDataset(os.path.join(DIR, '/content/drive/MyDrive/nlp-project /newtrain.csv'))
train_loader = DataLoader(train_set, batch_size=BATCH_SIZE, shuffle=True)
valid_set = QGDataset(os.path.join(DIR, '/content/drive/MyDrive/nlp-project /newvalid.csv'))
valid_loader = DataLoader(valid_set, batch_size=BATCH_SIZE, shuffle=False)
```

VALIDATION-LOOP:-



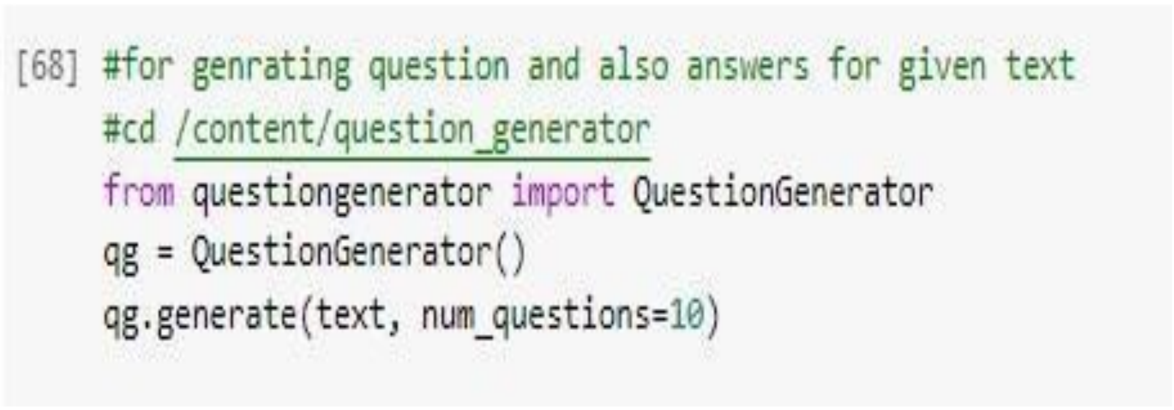
```
def evaluate(eval_model, data_loader):
    eval_model.eval()
    total_loss = 0.
    with torch.no_grad():
        for batch_index, batch in enumerate(data_loader):
            data, target = batch
            masked_labels = mask_label_padding(target['input_ids'])
            output = eval_model(
                input_ids=data['input_ids'],
                attention_mask=data['attention_mask'],
                labels=masked_labels
            )
            total_loss += output[0].item()
    return total_loss / len(data_loader)
```

TRAINING-LOOP:-



```
def train(epoch, best_val_loss):
    model.train()
    total_loss = 0.
    for batch_index, batch in enumerate(train_loader):
        data, target = batch
        optimizer.zero_grad()
        masked_labels = mask_label_padding(target['input_ids'])
        output = model(
            input_ids=data['input_ids'],
            attention_mask=data['attention_mask'],
            labels=masked_labels
        )
        loss = output[0]
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), 0.5)
        optimizer.step()

        total_loss += loss.item()
        if batch_index % LOG_INTERVAL == 0 and batch_index > 0:
            cur_loss = total_loss / LOG_INTERVAL
            print('| epoch {:3d} | '
                  '{:5d}/{:5d} batches | '
                  'loss {:5.2f}'.format(
                    epoch,
                    batch_index, len(train_loader),
                    cur_loss))
            save(
                TEMP_SAVE_PATH,
                epoch,
                model.state_dict(),
                optimizer.state_dict(),
                best_val_loss
            )
            total_loss = 0
```

FOR-ANSWER-GENERATION:-



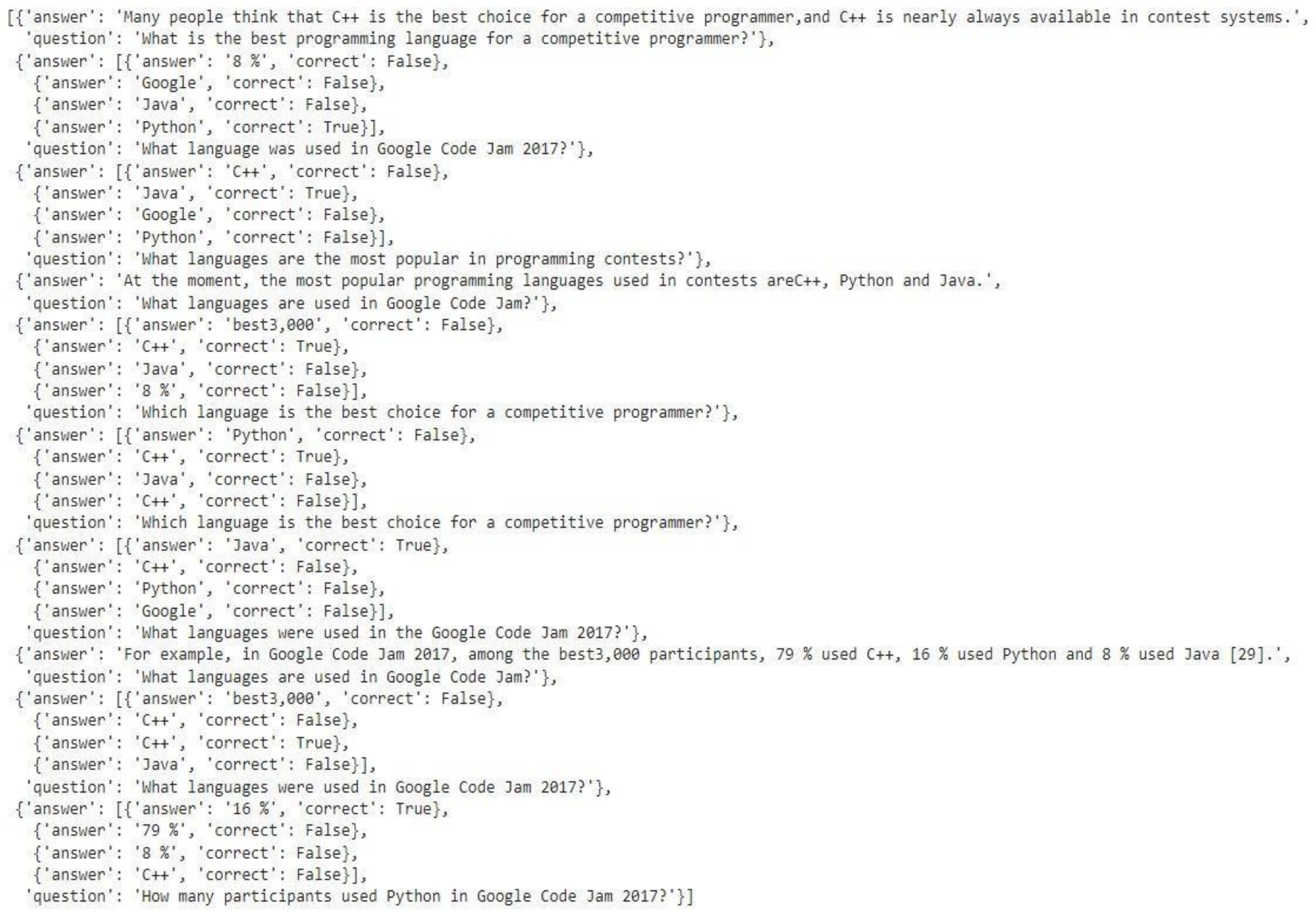
```
[68] #for genrating question and also answers for given text
     #cd /content/question_generator
     from questiongenerator import QuestionGenerator
     qg = QuestionGenerator()
     qg.generate(text, num_questions=10)
```

RESULT ON GIVING TEXT:-

TEXT:-At the moment, the most popular programming languages used in contests areC++, Python and Java. For example, in Google Code Jam

2017, among the best3,000 participants, 79 % used C++, 16 % used Python and 8 % used Java [29].Some participants also used several languages.Many people think that C++ is the best choice for a competitive programmer,and C++ is nearly always available in contest systems.

RESULT:-



```
[{'answer': 'Many people think that C++ is the best choice for a competitive programmer,and C++ is nearly always available in contest systems.',
  'question': 'What is the best programming language for a competitive programmer?'},
 {'answer': [{'answer': '8 %', 'correct': False},
   {'answer': 'Google', 'correct': False},
   {'answer': 'Java', 'correct': False},
   {'answer': 'Python', 'correct': True}],
  'question': 'What language was used in Google Code Jam 2017?'},
 {'answer': [{'answer': 'C++', 'correct': False},
   {'answer': 'Java', 'correct': True},
   {'answer': 'Google', 'correct': False},
   {'answer': 'Python', 'correct': False}],
  'question': 'What languages are the most popular in programming contests?'},
 {'answer': 'At the moment, the most popular programming languages used in contests areC++, Python and Java.',
  'question': 'What languages are used in Google Code Jam?'},
 {'answer': [{'answer': 'best3,000', 'correct': False},
   {'answer': 'C++', 'correct': True},
   {'answer': 'Java', 'correct': False},
   {'answer': '8 %', 'correct': False}],
  'question': 'Which language is the best choice for a competitive programmer?'},
 {'answer': [{'answer': 'Python', 'correct': False},
   {'answer': 'C++', 'correct': True},
   {'answer': 'Java', 'correct': False},
   {'answer': 'C++', 'correct': False}],
  'question': 'Which language is the best choice for a competitive programmer?'},
 {'answer': [{'answer': 'Java', 'correct': True},
   {'answer': 'C++', 'correct': False},
   {'answer': 'Python', 'correct': False},
   {'answer': 'Google', 'correct': False}],
  'question': 'What languages were used in the Google Code Jam 2017?'},
 {'answer': 'For example, in Google Code Jam 2017, among the best3,000 participants, 79 % used C++, 16 % used Python and 8 % used Java [29].',
  'question': 'What languages are used in Google Code Jam?'},
 {'answer': [{'answer': 'best3,000', 'correct': False},
   {'answer': 'C++', 'correct': False},
   {'answer': 'C++', 'correct': True},
   {'answer': 'Java', 'correct': False}],
  'question': 'What languages were used in Google Code Jam 2017?'},
 {'answer': [{'answer': '16 %', 'correct': True},
   {'answer': '79 %', 'correct': False},
   {'answer': '8 %', 'correct': False},
   {'answer': 'C++', 'correct': False}],
  'question': 'How many participants used Python in Google Code Jam 2017?'}]
```

______________________________END______________________________