Solving Probability and Statistics Problems by Program Synthesis

Leonard Tang
Harvard University
Mathematics

Elizabeth Ke
MIT
Mathematics

Nikhil Singh
MIT
Media Lab

Nakul Verma
Columbia University
Computer Science Department

Iddo Drori
MIT
EECS

Abstract

We solve university level probability and statistics questions by program synthesis using OpenAI's Codex, a Transformer trained on text and fine-tuned on code. We transform course problems from MIT's 18.05 Introduction to Probability and Statistics and Harvard's STAT110 Probability into programming tasks. We then execute the generated code to get a solution. Since these course questions are grounded in probability, we often aim to have Codex generate probabilistic programs that simulate a large number of probabilistic dependencies to compute its solution. Our approach requires prompt engineering to transform the question from its original form to an explicit, tractable form that results in a correct program and solution. To estimate the amount of work needed to translate an original question into its tractable form, we measure the similarity between original and transformed questions. Our work is the first to introduce a new dataset of university-level probability and statistics problems and solve these problems in a scalable fashion using the program synthesis capabilities of large language models.

1 Introduction

Let's say we play a game where I keep flipping a coin until I get heads. If the first time I get heads is on the n-th coin, then I pay you $2n-1$ dollars. How much would you pay me to play this game?

How would one solve this problem in an automated fashion? Existing approaches to solving such a problem, typical in university level probability and statistics courses, overwhelmingly hinge upon directing foundation models to formulate answers in a *deductive* fashion, whether via a sequence of steps (Hendrycks et al., 2021) or formal operations (Amini et al., 2019).

An alternate compelling approach is to simulate a given task on a large scale and aggregate results across multiple scenarios. Such an approach, usually dubbed as probabilistic programming in the literature (Wingate et al., 2011), offers a flexible mechanism for solving a variety of probabilistic tasks.

Inspired by this insight, our goal is to solve probability problems both via simulation and direct methods by leveraging the power of a program synthesizer such as OpenAI's Codex (Chen et al., 2021). Codex is a Transformer model trained on text and fine-tuned on code, which has the capacity to write programs that can simulate arbitrary stochastic tasks. Our core approach is neatly demonstrated in Figure 1. We transform raw question text into a programming task. This is then fed into Codex, generating a probabilistic program. We can then execute this code to get the correct response.

To the best of our knowledge, we are the first to propose such a simulation based approach to solve probability questions. To evaluate the efficacy of our approach, we collect two sets of 20 undergraduate-level probability and statistics problems, curated from MIT's 18.05 and Harvard's STAT110.

The key to our success lies in the carefully engineered prompts we present to Codex. Critically, we introduce the notion of *Concept-Grounded Task* prompting, i.e. priming Codex with related concepts and problem-solving strategies in its prompt (cf. Figure 2).

1.1 Related Work

Foundation models. Foundation models (Bommasani et al., 2021) such as GPT-3 (Brown et al., 2020) have demonstrated impressive and unforeseen emergent capabilities from their learning process, including aptitude in automatic speech recognition, vision, commonsense reasoning, and more (Moritz et al., 2020; Dosovitskiy et al., 2021; Bosselut et al., 2019). For the task of answering questions specifically, such models have recently achieved strong performance (Rajpurkar et al., 2018). How-

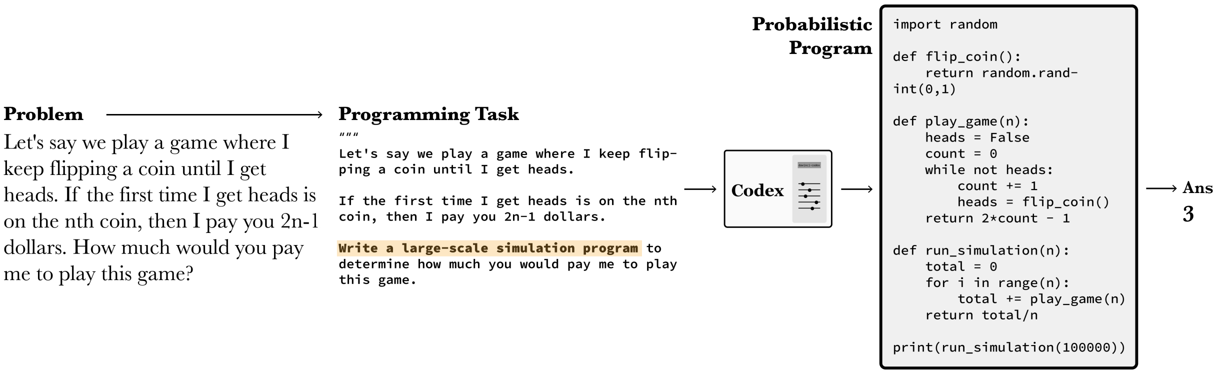

Figure 1: Probabilistic Simulation Program Workflow Example: (i) The original problem is translated into a programming task that asks Codex to simulate a large number of probabilistic scenarios, (ii) Codex generates such a program, and (iii) the program is executed to yield an answer.

ever, when tasked with solving university-level quantitative problems, foundation model performance is poor (Hendrycks et al., 2021).

Probability benchmarks. Though recent works have introduced datasets, such as MATH, MAWPS, MathQA, Math23k, and GSM8K (Hendrycks et al., 2021; Koncel-Kedziorski et al., 2016; Amini et al., 2019; Wang et al., 2017; Cobbe et al., 2021), that focus on benchmarking mathematical question answering, including probability questions, but all of these works only consider grade-school level question difficulty. We are the first to present two datasets of undergraduate-level probability and statistics questions.

2 Dataset

We introduce two datasets of questions from two separate undergraduate-level probability and statistics courses of varying difficulty and a set of quantitative finance interview questions. We describe the datasets below:

1. The first dataset consists of applied questions in probability. We take 20 questions that have numerical answers from MIT's 18.05: Introduction to Probability and Statistics (Bloom, 2014). Course topics include various probability and statistics concepts, including counting, conditional probability, discrete and continuous random variables, expectation and variance, central limit theorem, joint distributions, maximum likelihood estimators, Bayesian updating, null hypothesis significance testing, and confidence intervals.
2. The second dataset, in contrast to the first dataset, consists of conceptual questions in probability and statistics. We take 20 questions that have numerical answers from Harvard's STAT110: Probability (Blitzstein, 2021) and Brainstellar (Seth, 2021) online catalogue of quantitative finance probability brainteasers. Topics include: distributions, moment generating functions, expectation, variance, covariance, correlation, conditional probability, joint distributions, marginal distributions, conditional distributions, limit theorems, and Markov chains.

See Appendix for the list of all questions in our datasets.

3 Methods

3.1 Models and Evaluation

As our core program synthesizer, we leverage OpenAI's Codex (Chen et al., 2021). Given a raw question text, we use the following experimentation pipeline: we convert each question into a programming task, prompt Codex with the task to get a programmatic solution, and execute the program generated by Codex, comparing the execution result to the ground truth solution.

A critical component of this workflow is *prompt engineering* (i.e. the conversion from question to programming task). We examine prompt engineering in further detail in Section 3.2 and measure the degree to which we manipulate the prompt in Section 4.2.

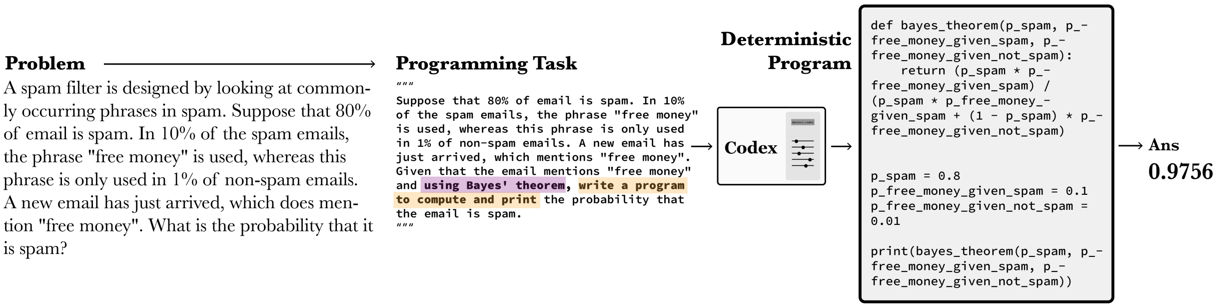

Figure 2: Concept-Grounded Workflow Example: (i) The original problem is translated into a programming task that includes Bayes' Theorem within its context, (ii) Codex generates a program, and (iii) the program is executed to yield an answer.

3.2 Prompt Engineering

Large generative language models, including Codex, are known to be extremely sensitive to input prompts (Reynolds and McDonell, 2021). Below, we outline and describe three classes of prompts that we communicate to Codex with and their associated effects:

- **Program Task Specification:** One class of prompts converts probability questions into direct task specifications (Reynolds and McDonell, 2021). In this case, these specifications are explicit programming assignments. For instance, if the original question is "What is the probability of flipping two heads in a row given a fair coin?", the corresponding task specification would be "Write a program that computes the probability of flipping two heads in a row given a fair coin." While these prompts occasionally suffice, in many instances additional prompt manipulation is required.

- **Probabilistic Simulation Programming**: While the above classes of prompts produce programs deterministic in nature, our third prompting technique hinges upon the power of probabilistic simulation programs, i.e. programs that simulate a large number of scenarios and aggregate results across simulations to determine an approximate answer. To trigger such simulation behavior in Codex, we include in our prompt the substring "Write a large-scale simulation program to estimate," followed by the desired task. Figure 1 presents an example using such a prompting scheme.

- **Concept-Grounded Task**: An extremely useful extension beyond *Program Task Specification* is to include relevant information pertaining to both the question and program contexts. The question context includes related topics or mathematical rules to use. This is primarily represented in the form of canonical equations, definitions, and theorems. Providing Codex with hints on problem-solving *strategy* involving these concepts is extremely helpful. For instance, if a question is related to the concept of Bayes' Theorem, appending the transformed prompt with an explicit instruction to use Bayes' Theorem results in a correct program. This is demonstrated in Figure 2. In addition to the question context, it may be useful to specify the programming context, including which packages or libraries the program will load and use, such as packages for symbolic math, integration, or optimization.

4 Results

4.1 Evaluating Program Output

It is possible for Codex to generate code which appears to yield a correct answer, but in reality is not a correct method to solve the problem. Thus, we explicitly inspect the code generated by Codex to check for its logical correctness. See Figure 3 for an explicit example which we encountered.

Another peculiar challenge in evaluating Codex's generated simulation programs lies in the approximate nature of their output. Specifically, we can only achieve perfect accuracy in the limit of simulation scale. Hence, we designate a numerical output resulting from program execution as correct when it is within 1% of the ground truth solution.

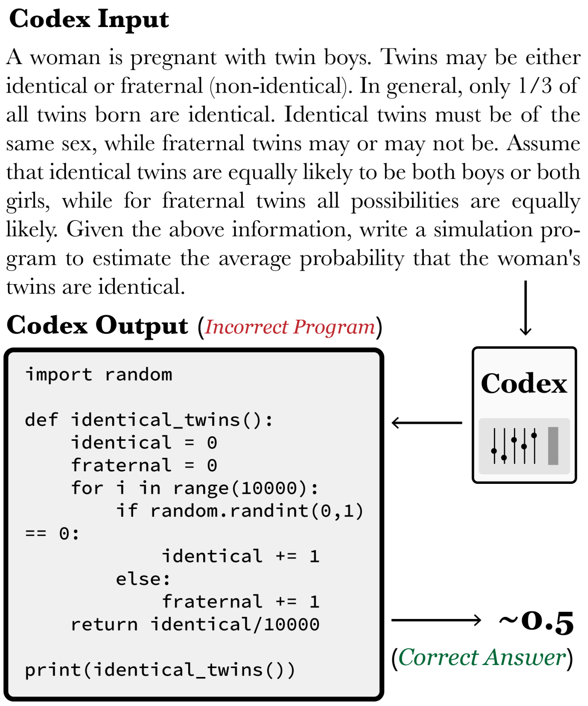

Figure 3: Evaluating Codex requires more than just checking the numerical answer. It requires evaluating the logic of the program, and seeing that it actually answers the program. Though the computed answer matches the ground truth solution of 0.5, the Codex program is written with the intention of calculating the unconditional probability of having fraternal twins (assuming fraternal and identical twins have equal probabilities), and completely ignores the conditioning information in the problem.

4.2 Achieving Perfect Results

Following the methods discussed in Section 3.2, we are able to generate correct programs for all questions in both datasets. See the Appendix for additional detail regarding the original question text, the corresponding prompt-engineered transformation, the generated program, and finally the program evaluation for each question in each dataset.

Since our approach involves prompt engineering in the process of translating a question to a programming task, we seek a concrete measure of the effort necessary in this transformation. Our metric is computed as follows: for any given pair of original question and transformed programming task, we compute the cosine similarity between the Sentence-BERT (Reimers and Gurevych, 2019) embedding of the question and Sentence-BERT embedding of the task.

Figure 4 shows that we have an average similarity of 0.80 in MIT's 18.05 and an average similarity of 0.79 in STAT110. As a baseline reference we also include the average pairwise similarity score among the original questions, thus indicating we only need minor changes to the text.

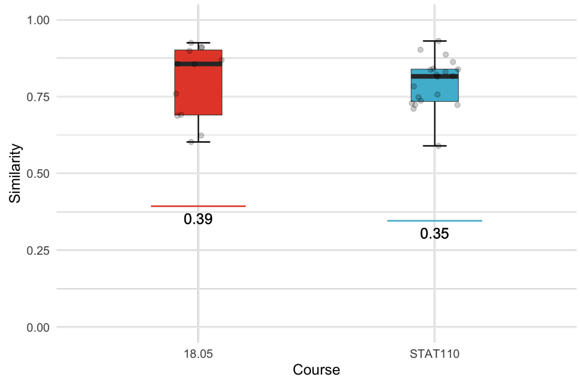

Figure 4: Sentence-BERT Similarity between original questions and programming tasks by course. Each course's transformation similarities are contextualized by baseline question similarities, i.e. the average pairwise similarity between the original questions.

Finally, since theses course are available online, we verify that Codex is not overfitting to training data, by writing and solving our own novel questions.

4.3 Implementation Details

We fix all Codex's hyperparameters to be the same for all experiments: *top-p*[1] is set to 1, sampling temperature is set to 0 (i.e. argmax), and maximum sequence length is set to 750 tokens. Both frequency and presence penalty are set to 0, and we do not halt on any stop sequences. We use the *davinci* engine for all of our generations.

5 Conclusion

To the best of our knowledge, this is the first work to present a state-of-the-art method that leverages the program synthesis and probabilistic simulation capabilities of foundation models to solve university-level probability and statistics problems. Through the use of prompt engineering, including priming Codex with concepts and problem-solving strategies, we achieve full correctness on our novel datasets. We plan to expand our work to further understand the underlying structure of programming tasks that are amenable to Codex manipulation, and in a similar vein move towards the automatic translation of questions to programming tasks.

[1] *top-p* designates the portion p of the token probability mass a language model samples from at each step.

References

Aida Amini, Saadia Gabriel, Shanchuan Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. 2019. MathQA: Towards interpretable math word problem solving with operation-based formalisms. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2357–2367, Minneapolis, Minnesota. Association for Computational Linguistics.

Joe Blitzstein. 2021. Statistics 110 probability.

Jonathan Bloom. 2014. Introduction to probability and statistics.

Rishi Bommasani et al. 2021. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*.

Antoine Bosselut, Hannah Rashkin, Maarten Sap, Chaitanya Malaviya, Asli Çelikyilmaz, and Yejin Choi. 2019. Comet: Commonsense transformers for automatic knowledge graph construction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Mark Chen et al. 2021. Evaluating large language models trained on code.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An image is worth 16x16 words: Transformers for image recognition at scale. *ICLR*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *NeurIPS*.

Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. 2016. MAWPS: A math word problem repository. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1152–1157, San Diego, California. Association for Computational Linguistics.

Niko Moritz, Takaaki Hori, and Jonathan Le. 2020. Streaming automatic speech recognition with the transformer model. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6074–6078.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789, Melbourne, Australia. Association for Computational Linguistics.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China. Association for Computational Linguistics.

Laria Reynolds and Kyle McDonell. 2021. Prompt programming for large language models: Beyond the few-shot paradigm. *CoRR*, abs/2102.07350.

Varun Seth. 2021. Brainstellar.

Yan Wang, Xiaojiang Liu, and Shuming Shi. 2017. Deep neural solver for math word problems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 845–854, Copenhagen, Denmark. Association for Computational Linguistics.

David Wingate, Andreas Stuhlmueller, and Noah Goodman. 2011. Lightweight implementations of probabilistic programming languages via transformational compilation. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 770–778, Fort Lauderdale, FL, USA. PMLR.

A MIT 18.05: Introduction to Probability and Statistics

<table>
<tr><td>Topic</td><td>Probability</td></tr>
<tr><td>Original question</td><td>In class we worked with non-transitive dice: Red: 3 3 3 3 3 6; Green: 1 4 4 4 4 4; White: 2 2 2 5 5 5. Find the probability that white beats green.</td></tr>
<tr><td>Codex input</td><td>Given non-transitive dice: Red: 3 3 3 3 3 6; Green: 1 4 4 4 4 4; White: 2 2 2 5 5 5, write a program that computes the probability that white beats green.</td></tr>
<tr><td>Codex output</td><td><pre>import itertools

red = [3, 3, 3, 3, 3, 6]
green = [1, 4, 4, 4, 4, 4]
white = [2, 2, 2, 5, 5, 5]

def prob_white_beats_green(red, green, white):
   rolls = list(itertools.product(red, green, white))
   wins = [1 for (r, g, w) in rolls if w > g]
   return sum(wins) / len(rolls)

prob_white_beats_green(red, green, white)</pre></td></tr>
<tr><td>Solution</td><td>7/12</td></tr>
</table>

Question 1 from 18.05

Topic	Probability
Original question	Calculate the probability of getting a two-pair poker hand.
Codex input	A hand is a set of 5 cards that are drawn randomly from a standard 52 card deck with 13 ranks of 4 cards each. A two-pair poker hand is a hand that contains 3 unique ranks, where no more than 2 cards in the hand can share the same rank. That is, 3 or more cards cannot share the same rank. Write a program that generates simulations for calculating the average probability of getting a two-pair poker hand.
Codex output	(see code below)
Solution	0.047539

Codex output:

```python
import random

def get_rank_counts(hand):
    """
    Returns a dictionary with ranks as keys and the number
        of cards with that rank as values.
    """
    rank_counts = {}
    for card in hand:
        rank = card[0]
        if rank in rank_counts:
            rank_counts[rank] += 1
        else:
            rank_counts[rank] = 1
    return rank_counts

def has_two_pair(hand):
    """
    Returns True if the hand has two pairs, and False
        otherwise.
    """
    rank_counts = get_rank_counts(hand)
    two_pair = False
    for rank in rank_counts:
        if rank_counts[rank] == 2:
            if two_pair:
                return True
            else:
                two_pair = True
    return False

def run_simulation():
    """
    Runs a simulation of drawing 5 cards randomly from a
        deck of 52 cards.
    """
    two_pair = 0
    for i in range(1000000):
        deck = [r+s for r in '23456789TJQKA' for s in 'SHDC']
        random.shuffle(deck)
        hand = deck[:5]
        if has_two_pair(hand):
            two_pair += 1
    return two_pair/1000000

print(run_simulation())
```

Question 2 from 18.05

Topic	Probability
Original question	Calculate the probability of getting a three-of-a-kind poker hand.
Codex input	A hand is a set of 5 cards that are drawn randomly from a standard 52 card deck with 13 ranks of 4 cards each. A three-pair poker hand is a hand that contains exactly 3 cards that share one rank, 1 card of another rank, and a final card of another rank. Write a program that generates simulations for calculating the average probability of getting a three-pair poker hand.
Codex output	See code below
Solution	0.021128

```python
import random

def get_rank_counts(hand):
    """
    Return a dictionary with ranks as keys and their counts
        as values.
    """
    ranks = [card[0] for card in hand]
    return {rank: ranks.count(rank) for rank in ranks}

def is_three_pair(hand):
    """
    Return True if hand is a three pair hand.
    """
    rank_counts = get_rank_counts(hand)
    return 3 in rank_counts.values()

def get_three_pair_probability(n):
    """
    Return the probability of getting a three pair hand.
    """
    three_pair_count = 0
    for _ in range(n):
        deck = [
            '2C', '2D', '2H', '2S',
            '3C', '3D', '3H', '3S',
            '4C', '4D', '4H', '4S',
            '5C', '5D', '5H', '5S',
            '6C', '6D', '6H', '6S',
            '7C', '7D', '7H', '7S',
            '8C', '8D', '8H', '8S',
            '9C', '9D', '9H', '9S',
            'TC', 'TD', 'TH', 'TS',
            'JC', 'JD', 'JH', 'JS',
            'QC', 'QD', 'QH', 'QS',
            'KC', 'KD', 'KH', 'KS',
            'AC', 'AD', 'AH', 'AS'
        ]
        hand = random.sample(deck, 5)
        if is_three_pair(hand):
            three_pair_count += 1
    return three_pair_count / n

print(get_three_pair_probability(1000000))
```

Question 3 from 18.05

<table>
<tr><td>Topic</td><td>Probability</td></tr>
<tr><td>Original question</td><td>In class we worked with non-transitive dice: Red: 3 3 3 3 3 6; Green: 1 4 4 4 4 4; White: 2 2 2 5 5 5. Find the probability that green beats red.</td></tr>
<tr><td>Codex input</td><td>Given two non-transitive dice:
Red: 3 3 3 3 3 6;
Green: 1 4 4 4 4 4.
Write a program that computes the probability that green beats red.</td></tr>
<tr><td>Codex output</td><td>

```python
def dice_prob(dice1, dice2):
    '''
    dice1: list of dice 1
    dice2: list of dice 2
    '''
    # Your code here
    red_wins = 0
    green_wins = 0
    for i in dice1:
        for j in dice2:
            if i > j:
                red_wins += 1
            elif j > i:
                green_wins += 1
    return green_wins / (red_wins + green_wins)

# Test Cases
dice1 = [3, 3, 3, 3, 3, 6]
dice2 = [1, 4, 4, 4, 4, 4]
print(dice_prob(dice1, dice2))
```

</td></tr>
<tr><td>Solution</td><td>25/36</td></tr>
</table>

Question 4 from 18.05

<table>
<tr><td>Topic</td><td>Conditional Probability</td></tr>
<tr><td>Original question</td><td>Mr. Smith has two children. At least one of them is a boy. What is the probability that both children are boys?</td></tr>
<tr><td>Codex input</td><td>Write a program that computes the probability that both children are boys given that one of them is a boy.</td></tr>
<tr><td>Codex output</td><td>

```python
# probability that one child is a boy
p_one_boy = 1/4

# probability that both children are boys
p_both_boys = 1/4

# probability that one of them is a boy
p_one_is_boy = 3/4

# probability that both children are boys given that one of
    them is a boy
p_both_boys_given_one_boy = p_both_boys / p_one_is_boy

print(p_both_boys_given_one_boy)
```

</td></tr>
<tr><td>Solution</td><td>1/3</td></tr>
</table>

Question 5 from 18.05

Topic	Discrete Random Variables, Standard Deviation
Original question	Let X be the result of rolling a fair 4-sided die. Let Y be the result of rolling a fair 6-sided die. Let Z be the average of X and Y. Find the standard deviation of X.
Codex input	In Probability, Write a program that computes the variance and standard deviation of rolling a fair 4-sided die 1000000 times.

Codex output	<pre>import random
	
	def variance(lst):
	    mean = sum(lst) / len(lst)
	    return sum((i - mean) ** 2 for i in lst) / len(lst)
	
	def standard_deviation(lst):
	    return variance(lst) ** 0.5
	
	def main():
	    lst = []
	    for i in range(1000000):
	        lst.append(random.randint(1, 4))
	    print(variance(lst))
	    print(standard_deviation(lst))
	
	if __name__ == '__main__':
	    main()</pre>

Solution	1.118

Question 6 from 18.05

Topic	Discrete Random Variables, Standard Deviation
Original question	Let X be the result of rolling a fair 4-sided die. Let Y be the result of rolling a fair 6-sided die. Let Z be the average of X and Y. Find the standard deviation of Y.
Codex input	In Probability, Write a program that computes the variance and standard deviation of rolling a fair 6-sided die 1000000 times.

Codex output	<pre>import statistics
	
	def variance_and_standard_deviation():
	    data = []
	    for i in range(1000000):
	        data.append(random.randint(1,6))
	    print(statistics.variance(data))
	    print(statistics.stdev(data))
	
	variance_and_standard_deviation()</pre>

Solution	1.708

Question 7 from 18.05

Topic	Discrete Random Variables, Standard Deviation
Original question	Let X be the result of rolling a fair 4-sided die. Let Y be the result of rolling a fair 6-sided die. Let Z be the average of X and Y. Find the standard deviation of Z.
Codex input	Let X be the result of rolling a fair 4-sided die. Let Y be the result of rolling a fair 6-sided die. Let Z be the average of X and Y. Write a program that computes the standard deviation of Z.

Codex output	`import math`
	
	`def standard_deviation(lst):`
	`    mean = sum(lst) / len(lst)`
	`    return math.sqrt(sum([(x - mean) ** 2 for x in lst]) /`
	`        len(lst))`
	
	`def main():`
	`    X = [1, 2, 3, 4]`
	`    Y = [1, 2, 3, 4, 5, 6]`
	`    Z = []`
	`    for i in X:`
	`        for j in Y:`
	`            Z.append((i + j) / 2)`
	`    print(standard_deviation(Z))`
	
	`if __name__ == "__main__":`
	`    main()`

Solution	1.021

Question 8 from 18.05

Topic	Discrete Random Variables, Expectation
Original question	Let X be the result of rolling a fair 4-sided die. Let Y be the result of rolling a fair 6-sided die. You win 2X dollars if X>Y and lose 1 dollar otherwise. After playing this game 60 times, what is your expected total gain?
Codex input	In Probability, X is the result of rolling a fair 4-sided die and Y is the result of rolling a fair 6-sided die. You win 2X if X > Y and lose 1 otherwise. Write a program that computes the expected total gain after rolling both dice 60 times.

Codex output	`import random`
	
	`def roll_dice():`
	`    return random.randint(1, 4)`
	
	`def roll_dice_6():`
	`    return random.randint(1, 6)`
	
	`def play_game():`
	`    total_gain = 0`
	`    for i in range(60):`
	`        x = roll_dice()`
	`        y = roll_dice_6()`
	`        if x > y:`
	`            total_gain += 2 * x`
	`        else:`
	`            total_gain -= 1`
	`    return total_gain`
	
	`def main():`
	`    total_gain = 0`
	`    for i in range(1000):`
	`        total_gain += play_game()`
	`    print(total_gain / 1000)`
	
	`if __name__ == '__main__':`
	`    main()`

Solution	55

Question 9 from 18.05

Topic	Counting, Continuous Variables
Original question	Boxes of Raisin Bran cereal are 30cm tall. Due to settling, boxes have a higher density of raisins at the bottom (h = 0) than at the top (h = 30). Suppose the density (in raisins per cm of height) is given by f(h) = 40 - h. How many raisins are in a box?
Codex input	Boxes of Raisin Bran cereal are 30cm tall. Suppose the density of raisins per cm of height is given by f(h) = 40 - h. Using scipy.integrate, write a program that computes how many raisins are in a box.
Codex output	(see code below)
Solution	750

```python
import numpy as np
import scipy.integrate as integrate

def f(h):
    return 40 - h

def integrate_f(a, b):
    return integrate.quad(f, a, b)

print(integrate_f(0, 30))
```

Question 10 from 18.05

Topic	Probability, Continuous Variables
Original question	Boxes of Raisin Bran cereal are 30cm tall. Due to settling, boxes have a higher density of raisins at the bottom (h = 0) than at the top (h = 30). Suppose the density (in raisins per cm of height) is given by f(h) = 40 - h. What is the probability that a random raisin is in the bottom third of the box?
Codex input	h ranges from 0 to 30. Integrate f(h) = 40 - h from 0 to 10. Divide by the integration of h from 0 to 30.
Codex output	(see code below)
Solution	7/15

```python
def f(h):
    return 40 - h

def integrate(f, a, b, N):
    """
    Integrates f from a to b using N intervals
    """
    dx = (b - a) / N
    total = 0
    for i in range(N):
        total += f(a + i * dx)
    return total * dx

print(integrate(f, 0, 10, 100000) / integrate(f, 0, 30,
    100000))
```

Question 11 from 18.05

Topic	Prior Odds
Original question	You have a drawer that contains 50 coins. 10 coins have probability p = 0.3 of heads, 30 coins have probability p = 0.5 and 10 coins have probability p = 0.7. You pick one coin at random from the drawer and flip it. What are the (prior) odds you chose a 0.3 coin?
Codex input	You have a drawer that contains 10 white coins and 40 black coins. You pick one coin at random from the drawer. Write a program that computes the odds that you chose a white coin. In probability, the odds of an event E is P(E)/P(E^c).

Codex output	`# P(E) = 10/50`
	`# P(E^c) = 40/50`
	`# odds = P(E)/P(E^c) = 10/40`
	
	`print(10/40)`

Solution	1/4

Question 12 from 18.05

Topic	Joint Distributions
Original question	Suppose X and Y have joint pdf $f(x, y) = c(x^2 + xy)$ on [0, 1] × [0, 1]. Find c.
Codex input	Define variables x, y, c. Using Sympy, integrate f(x, y) = c(x^2+xy) from 0 to 1 for x and 0 to 1 for y; set it equal to 1 in an equation. Find c that makes the expression 1.

Codex output	`import sympy as sp`
	
	`x, y, c = sp.symbols('x y c')`
	
	`f = c * (x**2 + x*y)`
	
	`sp.integrate(f, (x, 0, 1), (y, 0, 1))`
	
	`sp.solve(sp.Eq(1, sp.integrate(f, (x, 0, 1), (y, 0, 1))), c)`

Solution	12/7

Question 13 from 18.05

Topic	Central Limit Theorem, Probability
Original question	To head the newly formed US Dept. of Statistics, suppose that 50% of the population supports Erika, 20% supports Ruthi, and the rest are split between Peter, Jon and Jerry. A poll asks 400 random people who they support. Use the central limit theorem to estimate the probability that at least 52.5% of those polled prefer Erika?
Codex input	Suppose that 50% of the population supports Erika. A poll asks 400 random people who they support. Write a program that uses the central limit theorem to compute the probability that at least 52.5% of those polled support Erika.
Codex output	(see code below)
Solution	0.16

Codex output:

```
import random

def poll(n):
    """
    n: number of people polled
    returns: percentage of people in favor of Erika
    """
    # generate a list of n random numbers between 0 and 1
    # if the number is less than 0.5, the person is in favor
        of Erika
    # if the number is greater than 0.5, the person is in
        favor of Erica
    # return the percentage of people in favor of Erika
    count = 0
    for i in range(n):
        if random.random() < 0.5:
            count += 1
    return count / n

def poll_prob(n, m):
    """
    n: number of people polled
    m: number of times the poll is conducted
    returns: probability that at least 52.5% of the
        population supports Erika
    """
    # run poll_prob() m times
    # count the number of times the poll is in favor of Erika
    # return the number of times the poll is in favor of
        Erika divided by m
    count = 0
    for i in range(m):
        if poll(n) >= 0.525:
            count += 1
    return count / m

print(poll_prob(400, 10000))
```

Question 14 from 18.05

Topic	Central Limit Theorem, Probability
Original question	To head the newly formed US Dept. of Statistics, suppose that 50% of the population supports Erika, 20% supports Ruthi, and the rest are split between Peter, Jon and Jerry. A poll asks 400 random people who they support. Use the central limit theorem to estimate the probability that less than 25% of those polled prefer Peter, Jon or Jerry?
Codex input	Suppose that 30% of the population supports Peter. A poll asks 400 random people who they support. Write a program that uses the central limit theorem to compute the probability that less than 25% of those polled support Peter.
Codex output	(see code below)
Solution	0.0145

```python
import random

def poll(n, p):
   count = 0
   for i in range(n):
      if random.random() < p:
         count += 1
   return count

def main():
   peter_support = 0.3
   n = 400
   p = 0.25
   poll_size = 1000
   count = 0
   for i in range(poll_size):
      if poll(n, peter_support) < p * n:
         count += 1
   print(count / poll_size)

if __name__ == "__main__":
   main()
```

Question 15 from 18.05

Topic	Uniform Random Variables, Probability
Original question	Alice and Bob are trying to meet for lunch and both will arrive, independently of each other, uniformly and at random between noon and 1pm. Let A and B be the number of minutes after noon at which Alice and Bob arrive, respectively. Then A and B are independent uniformly distributed random variables on [0, 60]. Find the probability that Alice arrives before 12:30.
Codex input	A and B are trying to meet for lunch between 12:00 and 1:00. Alice and Bob arriving for lunch are independent and uniformly distributed events. Write a function that finds the probability that A arrives before 12:30.
Codex output	(see code below)
Solution	1/2

```python
def prob_a_before_1230(a, b):
   return a / (a + b)

print(prob_a_before_1230(0.5, 0.5))
```

Question 16 from 18.05

Topic	Uniform Random Variables, Probability
Original question	Alice and Bob are trying to meet for lunch and both will arrive, independently of each other, uniformly and at random between noon and 1pm. Let A and B be the number of minutes after noon at which Alice and Bob arrive, respectively. Then A and B are independent uniformly distributed random variables on [0, 60]. Now suppose that Alice and Bob are both rather impatient and will leave if they have to wait more than 15 minutes for the other to arrive. What is the probability that Alice and Bob will have lunch together?
Codex input	Alice and Bob are trying to meet for lunch and both will arrive, independently of each other, uniformly and at random between 12pm and 1pm. Suppose that Alice and Bob are both rather impatient and will leave if they have to wait more than 15 minutes for the other to arrive. Write a function that randomly computes the probability that Alice and Bob will have lunch together.

Codex output	`import random`
	
	`def lunch_together():`
	`    # Alice arrives at time`
	`    time_a = random.uniform(0, 60)`
	`    # Bob arrives at time`
	`    time_b = random.uniform(0, 60)`
	`    # If both arrive within 15 minutes of each other, they have lunch together`
	`    if abs(time_a - time_b) <= 15:`
	`        return True`
	`    else:`
	`        return False`
	
	`def prob_lunch_together():`
	`    # Run lunch_together() 10,000 times`
	`    count = 0`
	`    for i in range(10000):`
	`        if lunch_together():`
	`            count += 1`
	`    # Return probability`
	`    return count / 10000`
	
	`print(prob_lunch_together())`

Solution	7/16

Question 17 from 18.05

Topic	Prior Odds
Original question	You have a drawer that contains 50 coins. 10 coins have probability p = 0.3 of heads, 30 coins have probability p = 0.5 and 10 coins have probability p = 0.7. You pick one coin at random from the drawer and flip it. What are the (prior) odds you chose a 0.7 coin?
Codex input	You have a drawer that contains 10 white coins and 40 black coins. You pick one coin at random from the drawer. Write a program that computes the odds that you chose a white coin. In probability, the odds of an event E is P(E)/P(E^c).

Codex output	`# P(E) = 10/50`
	`# P(E^c) = 40/50`
	`# odds = P(E)/P(E^c) = 10/40`
	
	`print(10/40)`

Solution	1/4

Question 18 from 18.05

Topic	Null Hypothesis Significance Testing, Type I Error
Original question	One generates a number x from a uniform distribution on the interval $[0, \theta]$. One decides to test H0: $\theta = 2$ against HA: $\theta = 2$ by rejecting H0 if x ≤ 0.1 or x ≥ 1.9. Compute the probability of a type I error.
Codex input	One generates a number x from a uniform distribution on the interval [0, θ]. One decides to test H0: $\theta = 2$ against HA: $\theta = 2$ by rejecting H0 if x ≤ 0.1 or x ≥ 1.9. Using numpy, write a program that computes the probability of a type I error.
Codex output	(see code below)
Solution	0.1

Codex output:

```python
import numpy as np

theta = 2
n = 100000

x = np.random.uniform(0, theta, n)

type_I_error = np.sum(x <= 0.1) + np.sum(x >= 1.9)

print(type_I_error / n)
```

Question 19 from 18.05

Topic	Null Hypothesis Significance Testing, Type II Error
Original question	One generates a number x from a uniform distribution on the interval $[0, \theta]$. One decides to test H0: $\theta = 2$ against HA: $\theta = 2$ by rejecting H0 if x ≤ 0.1 or x ≥ 1.9. Compute the probability of a type II error if the true value of θ is 2.5.
Codex input	One generates a number x from a uniform distribution on the interval [0, θ]. One decides to test H0 : $\theta = 2$ against HA : $\theta = 2$ by rejecting H0 if x ≤ 0.1 or x ≥ 1.9, and accepted otherwise. Given that $\theta = 2.5$, write a program that computes the type II error. The type II error is the probability that the null hypothesis is mistakenly accepted.
Codex output	(see code below)
Solution	0.72

Codex output:

```python
import numpy as np

theta = 2.5

def type_II_error(theta, x):
    return 1 - np.sum(np.logical_or(x < 0.1, x > 1.9)) /
        len(x)

x = np.random.uniform(0, theta, 100000)

print(type_II_error(theta, x))
```

Question 20 from 18.05

B Harvard STAT110: Probability

<table>
<tr><td>Topic</td><td>Complementary Probability</td></tr>
<tr><td>Original Question</td><td>In any 15-minute interval, there is a 20% probability that you will see at least one shooting star. What is the probability that you see at least one shooting star in the period of an hour?</td></tr>
<tr><td>Codex Input</td><td>In any 15-minute interval, there is a 20% probability that you will see at least one shooting star.

There are 4 15-minute intervals in an hour.

Write a simulation program to estimate the probability that you see at least one shooting star in the period of an hour.</td></tr>
<tr><td>Codex Output</td><td><pre>
import random

def star():
    return random.random() <= 0.2

def star_in_hour():
    count = 0
    for i in range(4):
        if star():
            count += 1
    return count >= 1

def estimate_probability():
    count = 0
    for i in range(10000):
        if star_in_hour():
            count += 1
    return count / 10000

print(estimate_probability())
</pre></td></tr>
<tr><td>Solution</td><td>0.594</td></tr>
</table>

Question 1 from STAT110

Topic	Conditional Probability
Original Question	A certain couple tells you that they have two children, at least one of which is a girl. What is the probability that they have two girls?
Codex Input	A couple has two children, with at least one child being a girl. Write a simulation program to estimate the probability that the both children are girls.
Codex Output	(see code below)
Solution	1/3

```
import random

def random_kid():
    return random.choice([""boy"", ""girl""])

both_girls = 0
older_girl = 0
either_girl = 0

random.seed(0)
for _ in range(10000):
    younger = random_kid()
    older = random_kid()
    if older == ""girl"":
        older_girl += 1
    if older == ""girl"" and younger == ""girl"":
        both_girls += 1
    if older == ""girl"" or younger == ""girl"":
        either_girl += 1

print(""P(both | older):"", both_girls / older_girl)
print(""P(both | either):"", both_girls / either_girl)
```

Question 2 from STAT110

Topic	Conditional Probability
Original Question	You have a group of couples that decide to have children until they have their first girl, after which they stop having children. What is the expected gender ratio of the children that are born? What is the expected number of children each couple will have?
Codex Input	N couples decide to have children (both boys and girls) until they have their first girl, after which they stop having children. Write a sampling program to estimate the ratio of boys to girls after all N couples stop having children.

Codex Output	<pre>import random
	
	def children():
	    return random.choice(['boy', 'girl'])
	
	def children_until_girl(n):
	    boys = 0
	    girls = 0
	    for i in range(n):
	        child = children()
	        if child == 'boy':
	            boys += 1
	        else:
	            girls += 1
	            break
	    return (boys, girls)
	
	def children_until_girl_sampling(n, samples):
	    boys = 0
	    girls = 0
	    for i in range(samples):
	        child = children_until_girl(n)
	        boys += child[0]
	        girls += child[1]
	    return (boys, girls)
	
	def children_until_girl_ratio(n, samples):
	    boys, girls = children_until_girl_sampling(n, samples)
	    return float(boys) / float(girls)
	
	print(children_until_girl_ratio(5, 10000))</pre>

Solution	1/2

Question 3 from STAT110

<table>
<tr><td>Topic</td><td>Expected Value</td></tr>
<tr><td>Original Question</td><td>A lazy high school senior types up application and envelopes to n different colleges, but puts the applications randomly into the envelopes. What is the expected number of applications that went to the right college?</td></tr>
<tr><td>Codex Input</td><td>A lazy high school senior types up application and envelopes to n different colleges, but puts the applications randomly into the envelopes. Write a program to calculate the expected number of applications that were put in the right envelopes.</td></tr>
<tr><td>Codex Output</td><td>

```
import random

def main():
   n = int(input(""Enter the number of colleges: ""))
   num_trials = int(input(""Enter the number of trials: ""))
   count = 0
   for i in range(num_trials):
      applications = [i for i in range(n)]
      envelopes = [i for i in range(n)]
      random.shuffle(envelopes)
      for i in range(n):
         if applications[i] == envelopes[i]:
            count += 1
   print(""The expected number of applications that were
       put in the right envelopes is"", count/num_trials)

if __name__ == ""__main__"":
   main()
```

</td></tr>
<tr><td>Solution</td><td>1</td></tr>
</table>

Question 4 from STAT110

<table>
<tr><td>Topic</td><td>Expected Value and Conditional Expectation</td></tr>
<tr><td>Original Question</td><td>What's the expected number of coin flips until you get two heads in a row? What's the expected number of coin flips until you get two tails in a row?</td></tr>
<tr><td>Codex Input</td><td>Generate 1000 sequences of random coin flips, and end each sequence once you get two consecutive heads. Write a program that computes the average sequence length.</td></tr>
<tr><td>Codex Output</td><td>

```
import random

def flip_coin():
   return random.choice(['H', 'T'])

def flip_coin_until_consecutive_heads():
   seq = ''
   while True:
      seq += flip_coin()
      if seq[-2:] == 'HH':
         break
   return seq

def flip_coin_until_consecutive_heads_1000_times():
   seqs = []
   for i in range(1000):
      seqs.append(flip_coin_until_consecutive_heads())
   return seqs

def compute_average_sequence_length():
   seqs = flip_coin_until_consecutive_heads_1000_times()
   return sum([len(seq) for seq in seqs]) / len(seqs)

print(compute_average_sequence_length())
```

</td></tr>
<tr><td>Solution</td><td>6</td></tr>
</table>

Question 5 from STAT110

Topic	Expected Value
Original Question	Let's say we play a game where I keep flipping a coin until I get heads. If the first time I get heads is on the nth coin, then I pay you 2n-1 dollars. How much would you pay me to play this game?
Codex Input	Let's say we play a game where I keep flipping a coin until I get heads. If the first time I get heads is on the nth coin, then I pay you 2n-1 dollars. Write a large-scale simulation program to determine how much you would pay me to play this game.

Codex Output	`import random`
	
	`def flip_coin():`
	`    return random.randint(0,1)`
	
	`def play_game(n):`
	`    heads = False`
	`    count = 0`
	`    while not heads:`
	`        count += 1`
	`        heads = flip_coin()`
	`    return 2*count - 1`
	
	`def run_simulation(n):`
	`    total = 0`
	`    for i in range(n):`
	`        total += play_game(n)`
	`    return total/n`
	
	`print(run_simulation(100000))`

Solution	3

Question 6 from STAT110

Topic	Bayes' Theorem and Conditional Probability
Original Question	You have two coins, one of which is fair and comes up heads with a probability 1/2, and the other which is biased and comes up heads with probability 3/4. You randomly pick coin and flip it twice, and get heads both times. What is the probability that you picked the fair coin?
Codex Input	You have one coin that is fair and comes up heads with probability 1/2. You have another coin that is biased and comes up heads with probability 3/4. You pick either the fair or biased coin with 50-50 probability. You flip the chosen coin twice. Write a program using Bayes' Theorem to compute and print the probability of having picked the fair coin, given the fact that the chosen coin lands heads both times.

Codex Output	`# P(fair|H,H) = P(H,H|fair) * P(fair) / P(H,H)`
	`# P(H,H|fair) = P(H|fair) * P(H|fair)`
	`# P(H,H) = P(H|fair) * P(H|fair) * P(fair) + P(H|biased) *`
	`    P(H|biased) * P(biased)`
	`# P(fair|H,H) = (1/2 * 1/2 * 1/2) / (1/2 * 1/2 * 1/2 + 3/4`
	`    * 3/4 * 1/2)`
	
	`print((1/2 * 1/2 * 1/2) / (1/2 * 1/2 * 1/2 + 3/4 * 3/4 *`
	`    1/2))`

Solution	4/13

Question 7 from STAT110

Topic	Conditional Probability
Original Question	How many people do we need in a class to make the probability that two people have the same birthday more than 1/2? For simplicity, assume there are 365 days in a year.
Codex Input	Write a program to compute how many people do we need in a class to make the probability that two people have the same birthday more than 1/2? There are 365 days in a year.

Codex Output	`def birthday_paradox(n):`
	`    """`
	`    n: number of people in the class`
	`    """`
	`    return 1 - (1 - 1/365)**(n*(n-1)/2)`
	
	`print(birthday_paradox(23))`

Solution	23

Question 8 from STAT110

Topic	
Original Question	A bag contains one marble which is either green or blue, with equal probabilities. A green marble is put in the bag (so there are 2 marbles now), and then a random marble is taken out. The marble taken out is green. What is the probability that the remaining marble is also green?
Codex Input	A woman is pregnant with twin boys. Twins may be either identical or fraternal (non-identical). In general, only 1/3 of all twins born are identical. Identical twins must be of the same sex, while fraternal twins may or may not be. Assume that identical twins are equally likely to be both boys or both girls, while for fraternal twins all possibilities are equally likely. Given the above information, write a simulation program to estimate the average probability that the woman's twins are identical.

Codex Output	`import random`
	
	`def green_marble_probability(num_trials):`
	`    """`
	`    num_trials: number of trials`
	`    """`
	`    num_green = 0`
	`    for i in range(num_trials):`
	`        bag = ['green', 'blue']`
	`        random.shuffle(bag)`
	`        bag.append('green')`
	`        random.shuffle(bag)`
	`        if bag[0] == 'green':`
	`            num_green += 1`
	`    return num_green / num_trials`
	
	`print(green_marble_probability(100000))`

Solution	2/3

Question 9 from STAT110

<table>
<tr><td>Topic</td><td>Conditional Probability</td></tr>
<tr><td>Original Question</td><td>Calvin and Hobbes play a match consisting of a series of games, where Calvin has probability 0.5 of winning each game (independently). They play with a 'win by two' rule: the first player to win two games more than his opponent wins the match. Find the probability that Calvin wins the match (in terms of p)</td></tr>
<tr><td>Codex Input</td><td>Calvin and Hobbes play a match consisting of a series of games, where Calvin has probability 0.5 of winning each game (independently).

They play with a 'win by two' rule: the first player to win two games more than his opponent wins the match.

Write a simulation program to estimate the average probability that Calvin wins the match.</td></tr>
<tr><td>Codex Output</td><td>

```python
import random

def calvin_wins(n):
    wins = 0
    for i in range(n):
        if random.random() < 0.5:
            wins += 1
    return wins >= 2

def calvin_wins_match(n):
    wins = 0
    for i in range(n):
        if calvin_wins(2):
            wins += 1
    return wins / n

print(calvin_wins_match(100000))
```

</td></tr>
<tr><td>Solution</td><td>0.25</td></tr>
</table>

Question 10 from STAT110

<table>
<tr><td>Topic</td><td>Geometric Probability</td></tr>
<tr><td>Original Question</td><td>Two witches make a nightly visit to an all-night coffee shop. Each arrives at a random time between 0:00 and 1:00. Each one of them stays for exactly 30 minutes. On any one given night, what is the probability that the witches will meet at the coffee shop?</td></tr>
<tr><td>Codex Input</td><td>Two witches make a nightly visit to an all-night coffee shop.
Each arrives at a random time between 0:00 and 1:00.
Each witch stays at the coffee shop for exactly 30 minutes.
Write a simulation program to estimate the average probability that the witches will see each other at the coffee shop.</td></tr>
<tr><td>Codex Output</td><td><pre>
import random

def witches():
   witches = []
   for i in range(2):
      witches.append(random.randint(0,60))
   return witches

def main():
   witches_seen = 0
   for i in range(100000):
      w = witches()
      if w[0] > w[1]:
         if w[0] - w[1] < 30:
            witches_seen += 1
      else:
         if w[1] - w[0] < 30:
            witches_seen += 1
   print(witches_seen/100000)

main()
</pre></td></tr>
<tr><td>Solution</td><td>0.75</td></tr>
</table>

Question 11 from STAT110

<table>
<tr><td>Topic</td><td>Geometric Probability</td></tr>
<tr><td>Original Question</td><td>p and q are two points chosen at random between 0 & 1. What is the probability that the ratio p/q lies between 1 & 2?</td></tr>
<tr><td>Codex Input</td><td>Roll a die.
If the die is 4, 5, or 6, you get paid the value of the roll.
If the die is 1, 2, or 3, you reroll and get paid the value of the new roll.
Write a program that computes the expected value.</td></tr>
<tr><td>Codex Output</td><td><pre>
import random

def prob_p_q():
   p = random.random()
   q = random.random()
   if p/q >= 1 and p/q <= 2:
      return True
   else:
      return False

def prob_p_q_sim(n):
   count = 0
   for i in range(n):
      if prob_p_q():
         count += 1
   return count/n

print(prob_p_q_sim(100000))
</pre></td></tr>
<tr><td>Solution</td><td>1/4</td></tr>
</table>

Question 12 from STAT110

Topic	Conditional Expectation
Original Question	Roll a die, and you get paid what the dice shows. But if you want, you can request a second chance & roll the die again; get paid what the second roll shows instead of the first. What is the expected value?
Codex Input	Roll a die, and you get paid what the number on the dice. You can request a second chance and roll the die again. You get paid the number on second roll instead of the first. Write a function to compute the expected value in Monte Carlo.

Codex Output	`import random`
	
	`def roll_die():`
	`    return random.randint(1,6)`
	
	`def second_chance():`
	`    return random.randint(1,6)`
	
	`def monte_carlo(n):`
	`    total = 0`
	`    for i in range(n):`
	`        first_roll = roll_die()`
	`        second_roll = second_chance()`
	`        total += max(first_roll, second_roll)`
	`    return total/n`
	
	`print(monte_carlo(100000))`

Solution	4.25

Question 13 from STAT110

<table>
<tr><td>Topic</td><td>Conditional Expectation</td></tr>
<tr><td>Original Question</td><td>A very innocent monkey throws a fair die. The monkey will eat as many bananas as are shown on the die, from 1 to 5. But if the die shows ’6’, the monkey will eat 5 bananas and throw the die again. This may continue indefinitely. What is the expected number of bananas the monkey will eat?</td></tr>
<tr><td>Codex Input</td><td>A very innocent monkey throws a fair 6-sided die repeatedly.
If the die value is 1 to 5, the monkey eats that many bananas and stops.
If the die value is 6, the monkey will eat 5 bananas.
Write a program that computes the average number of bananas that the monkey will eat before it stops.</td></tr>
<tr><td>Codex Output</td><td><pre><code>import random

def main():
    # ask user for number of trials
    trials = int(input(""How many trials? ""))
    # initialize the number of bananas eaten
    bananas = 0
    # loop for the number of trials
    for i in range(trials):
        # initialize the number of bananas eaten
        bananas_eaten = 0
        # loop until the monkey stops eating bananas
        while True:
            # generate a random number between 1 and 6
            die = random.randint(1, 6)
            # if the die value is 1 to 5, the monkey eats that many bananas and stops
            if die <= 5:
                bananas_eaten += die
                break
            # if the die value is 6, the monkey will eat 5 bananas
            elif die == 6:
                bananas_eaten += 5
        # add the number of bananas eaten to the total
        bananas += bananas_eaten
    # calculate the average number of bananas eaten
    average = bananas / trials
    # print the average number of bananas eaten
    print(""The average number of bananas eaten is"", average)

main()</code></pre></td></tr>
<tr><td>Solution</td><td>4</td></tr>
</table>

Question 14 from STAT110

Topic	Conditional Expectation
Original Question	A fair coin is tossed repeatedly until 5 consecutive heads occurs. What is the expected number of coin tosses?
Codex Input	Write a program to simulate the average number of coin flips needed until you get 5 heads in a row for the first time.

Codex Output	```import random
	
	def flip_coin():
	    return random.choice(['H', 'T'])
	
	def flip_until_five_heads():
	    heads = 0
	    count = 0
	    while heads < 5:
	        count += 1
	        if flip_coin() == 'H':
	            heads += 1
	        else:
	            heads = 0
	    return count
	
	def average_flips_until_five_heads():
	    total = 0
	    for i in range(1000):
	        total += flip_until_five_heads()
	    return total / 1000
	
	print(average_flips_until_five_heads())```

Solution	62

Question 15 from STAT110

Topic	Geometric Probability, Expected Value
Original Question	A stick of length 1 breaks at a random point distributed uniformly across it length. What is the expected length of the smaller part of the broken stick?
Codex Input	A stick of length 1 is split into two portions at a random point distributed uniformly across it length. Write a simulation program that estimates the average length the shorter portion of the stick.
Codex Output	see code below
Solution	1/4

```
import random

def stick_splitting(n):
    """
    :param n: number of simulations to run
    :return: average length of the shorter portion of the
        stick after splitting it in half n times.
    """

    total = 0

    for i in range(n):

        # generate a random number between 0 and 1,
            representing the point at which to split the
            stick.
        split_point = random.random()

        # generate a random number between 0 and 1,
            representing the length of the first portion of
            the stick.
        first_portion = random.random()

        # generate a random number between 0 and 1,
            representing the length of the second portion of
            the stick.
        second_portion = 1 - first_portion

        # add up all lengths from each simulation run to get
            an average length for each simulation run.
        total += min(first_portion, second_portion)

    return total / n

print(stick_splitting(100000))
```

Question 16 from STAT110

Topic	Expected Value
Original Question	You are given an urn with 100 balls (50 black and 50 white). You pick balls from urn one by one without replacements until all the balls are out. A black followed by a white or a white followed by a black is 'a color change'. Calculate the expected number of colour changes if the balls are being picked randomly from the urn.
Codex Input	You are given an urn with 100 balls (50 black and 50 white). A sequence of balls is drawn from the urn without replacement until the urn is empty. Drawing a black ball immediately followed by a white ball is considered a color change. Drawing a white ball immediately followed by a black ball is also considered a color change. Write a simulation program that estimates the average number of color changes in a 100-ball sequence.
Codex Output	See code below
Solution	50

Codex Output:

```
import random

def color_change(urn):
    """"""Returns the number of color changes in a 100-ball
        sequence.""""""
    count = 0
    while len(urn) > 0:
        ball = urn.pop()
        if len(urn) == 0: # last ball, no color change
            break

        next_ball = urn[-1] # next ball in sequence

        if (ball == 'B' and next_ball == 'W') or (ball == 'W'
            and next_ball == 'B'): # color change!
            count += 1

    return count


def main():

    urn = ['B'] * 50 + ['W'] * 50 # create an urn with 50
        black and 50 white balls

    trials = 10000 # number of trials to run for simulation

    total_color_changes = 0 # initialize total number of
        color changes to zero

    for i in range(trials): # run the simulation for the
        specified number of trials
        random.shuffle(urn) # shuffle the balls in the urn
            before each trial

        total_color_changes += color_change(list(urn)) # add
            the number of color changes in this trial to the
            total

    print(total_color_changes / trials) # print the average
        number of color changes over all trials

if __name__ == '__main__':
    main()
```

Question 17 from STAT110

<table>
<tr><td>Topic</td><td>Conditional Probability</td></tr>
<tr><td>Original Question</td><td>X and Y are i.i.d. N(0, 1) random variables. You are given that $X > 0$ and $Y > 0$. What is the probability that $Y > X$?</td></tr>
<tr><td>Codex Input</td><td>X and Y are i.i.d. N(0, 1) random variables. Write a program to calculate the probability that $Y > X$ if both $X > 0$ and $Y > 0$.</td></tr>
<tr><td>Codex Output</td><td>

```python
# X and Y are i.i.d. N(0, 1) random variables.
# Write a program to calculate the probability that Y > X if both X > 0 and Y > 0.

import numpy as np
import matplotlib.pyplot as plt
from scipy import stats


def prob_Y_greater_X():

    # Generate 1000 samples of X and Y, where X ~ N(0, 1) and Y ~ N(0, 1).

    x = np.random.normal(loc=0, scale=1, size=1000)
    y = np.random.normal(loc=0, scale=1, size=1000)

    # Calculate the probability that Y > X if both X > 0 and Y > 0

    prob = sum([1 for i in range(len(x)) if x[i]>0 and y[i]>0 and y[i]>x[i]])/sum([1 for i in range(len(x)) if x[i]>0 and y[i]>0])

    return prob


if __name__ == '__main__':

    print(prob_Y_greater_X())
```

</td></tr>
<tr><td>Solution</td><td>1/2</td></tr>
</table>

Question 18 from STAT110

Topic	Conditional Expectation
Original Question	Suppose that you roll a dice. For reach roll, yo uare paid the face value. If a roll gives 4, 5, or 6, you can roll the dice again. If you get 1, 2, or 3, the game stops. What is the expected payoff of this game?
Codex Input	Suppose we play a game where you roll a dice. If a roll gives 4, 5, or 6, you are paid the face value and you roll the dice again. If a roll gives 1, 2, or 3, you are paid the face value and you stop rolling. Write a simulation program that estimates the average payoff of this game.
Codex Output	(see code below)
Solution	7

Codex Output:

```
import random

def roll_dice():
    return random.randint(1,6)

def play_game():
    total = 0.0
    while True:
        roll = roll_dice()
        if roll == 1 or roll == 2 or roll == 3: # stop
            rolling and get paid the face value of the dice
            roll.
            total += roll # add the face value of the dice to
                the total.
            break      # stop playing the game.
        else:           # keep rolling and add the face value
            to the total.
            total += roll

    return total     # return the total amount won from
        playing this game.


# main program starts here, DO NOT change it. It runs the
    simulation and prints results.
number_of_games = 10000 # number of times we play this
    game. Change it to 1000 to see more accurate results.
total = 0            # initial condition - we don't have any
    money at hand right now.
for i in range(number_of_games):
    total += play_game() # add the amount won from each game
        to get the total amount won.
print(""The average payoff from {} games is
    {}"".format(number_of_games, total/number_of_games))
```

Question 19 from STAT110

Topic	Bayes' Theorem and Conditional Probability
Original Question	A spam filter is designed by looking at commonly occurring phrases in spam. Suppose that 80% of email is spam. In 10% of the spam emails, the phrase "free money" is used, whereas this phrase is only used in 1% of non-spam emails. A new email has just arrived, which does mention "free money". What is the probability that it is spam?
Codex Input	Suppose that 80% of email is spam. In 10% of the spam emails, the phrase "free money" is used, whereas this phrase is only used in 1% of non-spam emails. A new email has just arrived, which mentions "free money". Given that the email mentions "free money" and using Bayes' theorem, write a program to compute and print the probability that the email is spam.

Codex Output	`def bayes_theorem(p_spam, p_free_money_given_spam, p_free_money_given_not_spam):`
	`return (p_spam * p_free_money_given_spam) / (p_spam * p_free_money_given_spam + (1 - p_spam) * p_free_money_given_not_spam)`
	
	`p_spam = 0.8`
	`p_free_money_given_spam = 0.1`
	`p_free_money_given_not_spam = 0.01`
	
	`print(bayes_theorem(p_spam, p_free_money_given_spam, p_free_money_given_not_spam))`

Solution	0.9756

Question 20 from STAT110