Slip 1-1

```python
import random

# Define the objective function (you can replace this with your own)
def objective_function(x):
    return -(x**2 + 4*x)  # Example: maximize x^2 + 4x (negate for maximization)

def hill_climbing(max_iterations, step_size):
  current_solution = random.uniform(-10, 10)  # Start with a random initial solution
  current_value = objective_function(current_solution)

  for _ in range(max_iterations):
    neighbor = current_solution + random.uniform(-step_size, step_size)
    neighbor_value = objective_function(neighbor)

    if neighbor_value>current_value:
      current_solution = neighbor
      current_value = neighbor_value
  return current_solution, current_value

if __name__ == "__main__":
  max_iterations = 1000  # Maximum number of iterations
  step_size = 0.1  # Step size for making small changes

  final_solution, final_value = hill_climbing(max_iterations, step_size)

  print("Final Solution:", final_solution)
  print("Objective Value at Final Solution:", final_value)

# https://www.mathway.com/Algebra --> refer for answer
```

Slip 2-1

```
# Import the 'calendar' module
import calendar

# Prompt the user to input the year and month
y = int(input("Input the year : "))
m = int(input("Input the month : "))

# Print the calendar for the specified year and month
print(calendar.month(y, m))
```

slip 3-1

```
# define punctuation
punctuations = '''!()-[]{};:'"\,<>./?@#$%^&*_~'''

my_str = "Hello!!!, he said ---and went."

# To take input from the user
# my_str = input("Enter a string: ")

# remove punctuation from the string
no_punct = ""
for char in my_str:
   if char not in punctuations:
       no_punct = no_punct + char
print(no_punct)
```

slip 4-1

```
#importing the time module
import time

#welcoming the user
name = input("What is your name? ")

print ("Hello, " + name, "Time to play hangman!")

#wait for 1 second
time.sleep(1)

print ("Start guessing...")
```

```
time.sleep(0.5)

#here we set the secret. You can select any word to play with.
word = ("secret")

#creates an variable with an empty value
guesses = ''

#determine the number of turns
turns = 10

# Create a while loop

#check if the turns are more than zero
while turns > 0:

    # make a counter that starts with zero
    failed = 0

    # for every character in secret_word
    for char in word:

    # see if the character is in the players guess
        if char in guesses:

        # print then out the character
            print (char,end=""),

        else:

        # if not found, print a dash
            print ("_",end=""),

        # and increase the failed counter with one
            failed += 1

    # if failed is equal to zero

    # print You Won
    if failed == 0:
        print ("You won")
    # exit the script
        break
    # ask the user go guess a character
    guess = input("guess a character:")

    # set the players guess to guesses
    guesses += guess
    # if the guess is not found in the secret word
    if guess not in word:

     # turns counter decreases with 1 (now 9)
        turns -= 1

    # print wrong
        print ("Wrong")

    # how many turns are left
        print ("You have", + turns, 'more guesses' )

    # if the turns are equal to zero
        if turns == 0:

        # print "You Lose"
            print ("You Lose"  )
```

slip 5-1

```
import nltk
from nltk.stem import   WordNetLemmatizer

wordnet_lemmatizer = WordNetLemmatizer()

text = "studies studying cries cry "
nltk.download('punkt')
nltk.download('wordnet')
tokenization = nltk.word_tokenize(text)
for w in tokenization:
  print("Lemma for {} is {}".format(w,
wordnet_lemmatizer.lemmatize(w)))
```

slip 6-1

```
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords
import nltk
nltk.download('stopwords')
nltk.download('punkt')
data = text = open("input6.txt").read().lower()
stopWords = set(stopwords.words('english'))
words = word_tokenize(data)
wordsFiltered = [w for w in words if w not in stopWords]

print(wordsFiltered)
```

slip 7-1

```
pip install easyAI
from easyAI import TwoPlayerGame
from easyAI.Player import Human_Player


class TicTacToe(TwoPlayerGame):
    """The board positions are numbered as follows:
    1 2 3
    4 5 6
    7 8 9
    """

    def __init__(self, players):
        self.players = players
        self.board = [0 for i in range(9)]
        self.current_player = 1  # player 1 starts.

    def possible_moves(self):
        return [i + 1 for i, e in enumerate(self.board) if e == 0]

    def make_move(self, move):
        self.board[int(move) - 1] = self.current_player

    def unmake_move(self, move):  # optional method (speeds up the AI)
        self.board[int(move) - 1] = 0

    def lose(self):
        """ Has the opponent "three in line ?" """
        return any(
            [
                all([(self.board[c - 1] == self.opponent_index) for c
in line])
                for line in [
                    [1, 2, 3],
                    [4, 5, 6],
                    [7, 8, 9],  # horiz.
                    [1, 4, 7],
                    [2, 5, 8],
                    [3, 6, 9],  # vertical
                    [1, 5, 9],
                    [3, 5, 7],
                ]
            ]
        )  # diagonal
    def is_over(self):
        return (self.possible_moves() == []) or self.lose()

    def show(self):
        print(
            "\n"
            + "\n".join(
                [
                    " ".join([[".", "O", "X"][self.board[3 * j + i]]
for i in range(3)])
                    for j in range(3)
                ]
            )
        )

    def scoring(self):
        return -100 if self.lose() else 0


if __name__ == "__main__":

    from easyAI import AI_Player, Negamax

    ai_algo = Negamax(6)
    TicTacToe([Human_Player(), AI_Player(ai_algo)]).play()
```

Slip 10-1

```
from itertools import permutations

def solve_cryptarithmetic(puzzle):
  unique_chars = set("".join(puzzle))
  if len(unique_chars) > 10:
    print("Too many unique characters for a valid puzzle.")
    return

  for perm in permutations('0123456789', len(unique_chars)):
    char_to_digit = dict(zip(unique_chars, perm))
    if char_to_digit[puzzle[0][0]] == '0' or
char_to_digit[puzzle[1][0]] == '0' or char_to_digit[puzzle[2][0]] ==
'0':
      continue

    expression = "".join([char_to_digit[char] for char in puzzle[0]]) +
"+" + \
    "".join([char_to_digit[char] for char in puzzle[1]]) + "==" + \
    "".join([char_to_digit[char] for char in puzzle[2]])
```

```python
    if eval(expression):
      print(f"Solution found: {expression}")

# Example puzzle: TWO + TWO = FOUR
puzzle = ["TWO", "TWO", "FOUR"]
solve_cryptarithmetic(puzzle)
```