In [ ]:

In [ ]:

Question 1

In [ ]:
```
#This 'a' value of 0 and it has a global scope.
#this b() has a global scope
#Initially, the global variable a is assigned the value 0.

#When the function b() is called for the first time: It updates the global variable
#Inside c(a), a is passed as an argument and then incremented by 2. So, a becomes 0

#When the function b() is called for the second time: It again updates the global v
#This time, a is now 2. Inside c(a), a is passed as an argument and then incremente

#When the function b() is called for the third time: It once more updates the globa
#Inside c(a), a is passed as an argument and then incremented by 2. So, a becomes 4

#Therefore, when the last expression a is evaluated, the value displayed is 6.
```

In [63]:
```
a = 0

def b():
    global a
    a = c(a)

def c(a):
    return a + 2


print("Initial value of a:", a)  # Output: Initial value of a: 0
print("\n")
b()  # Call function b
print("Value of a after calling b:", a)
```

```
Initial value of a: 0


Value of a after calling b: 2
```

In [ ]:

In [ ]:

Question 2

In [ ]:
```
Function fileLength(), given to you, takes the name of a file as input and returns
    >>> fileLength('midterm.py') 284 >>> fileLength('idterm.py')
    Traceback (most recent call last): File "<pyshell#34>", line 1, in <module> fil
        File "/Users/me/midterm.py", line 3, in fileLength infile = open(filename)
            [Errno 2] No such file or directory: 'idterm.py'
As shown above, if the file cannot be found by the interpreter or if it cannot be r
an exception will be raised. Modify function fileLength() so that a friendly messag
    >>> fileLength('midterm.py') 358 >>> fileLength('idterm.py') File idterm.py not
```

In [27]:
```
def fileLength(filename):
    try:
        with open(filename, 'r') as infile:
            content = infile.read()
        return len(content)
    except FileNotFoundError:
        print(f"File {filename} not found.")
        return

print(fileLength('midterm.py'))  # Output: Length of the file content if found
print(fileLength('idterm.py'))   # Output: File idterm.py not found.
print(fileLength(r"C:\Users\chikw\Downloads\Filelength (1).txt"))
```

```
File midterm.py not found.
None
File idterm.py not found.
None
123
```

In [ ]:

Question 3

In [ ]:
```
Write a class named Marsupial that can be used as shown below:
    >>> m = Marsupial()
    >>> m.put_in_pouch('doll')
    >>> m.put_in_pouch('firetruck')
    >>> m.put_in_pouch('kitten')
    >>> m.pouch_contents()
    ['doll', 'firetruck', 'kitten']
```

In [22]:
```
class Marsupial:
    def __init__(self):
        self.pouch = []

    def put_in_pouch(self, item):
        self.pouch.append(item)

    def pouch_contents(self):
        return self.pouch

# Test the Marsupial class
m = Marsupial()
m.put_in_pouch('doll')
m.put_in_pouch('firetruck')
m.put_in_pouch('kitten')
print(m.pouch_contents())

print("\n")
class Kangaroo(Marsupial):
    def __init__(self, x=0, y=0):
        super().__init__()
        self.x = x
        self.y = y

    def jump(self, dx, dy):
        self.x += dx
        self.y += dy

    def __str__(self):
        return f"I am a Kangaroo located at coordinates ({self.x},{self.y})"
```

```
# Test the Kangaroo class
k = Kangaroo(0, 0)
print(k)  # Output: I am a Kangaroo located at coordinates (0,0)

print("\n")
k.put_in_pouch('doll')
k.put_in_pouch('firetruck')
k.put_in_pouch('kitten')
print(k.pouch_contents())  # Output: ['doll', 'firetruck', 'kitten']

print("\n")
k.jump(1, 0)
k.jump(1, 0)
k.jump(1, 0)
print(k)  # Output: I am a Kangaroo located at coordinates (3,0)
```

```
['doll', 'firetruck', 'kitten']


I am a Kangaroo located at coordinates (0,0)


['doll', 'firetruck', 'kitten']


I am a Kangaroo located at coordinates (3,0)
```

In [ ]:

Question 4

In [ ]:
```
Write function collatz() that takes a positive integer x as input and prints the Co
A Collatz sequence is obtained by repeatedly applying this rule to the previous num
x = { x/2 if x is even3x+1 if x is oddYour function should stop when the sequence get
     Your implementation must be recursive, without any loops.
>>> collatz(1)
1
>>> collatz(10)
10
5
16
8
4
2
1
```

In [65]:
```
def collatz(x):
    print(x)
    if x == 1:
        return
    elif x % 2 == 0:
        collatz(x // 2)
    else:
        collatz(3 * x + 1)

collatz(1)
collatz(10)
```

```
1
10
5
16
8
4
2
1
```

In [ ]:

Question 5

In [4]:
```
def binary(n):
    if n < 2:
        print(n, end='')
    else:
        binary(n // 2)
        print(n % 2, end='')

# Test cases
binary(0)  # Output: 0
print()   # Newline for formatting
binary(1)  # Output: 1
print()   # Newline for formatting
binary(3)  # Output: 11
print()   # Newline for formatting
binary(9)  # Output: 1001
```

```
0
1
11
1001
```

Question 6

In [ ]:
```
Implement a class named HeadingParser that can be used to parse an HTML document, a
document. You should implement your class as a subclass of HTMLParser, defined in S
When fed a string containing HTML code, your class should print the headings, one p
they appear in the document. Each heading should be indented as follows: an h1 head
    h2 heading should have indentation 1, etc. Test your implementation using w3c.h
>>> infile = open('w3c.html')
>>> content = infile.read()
>>> infile.close()
>>> hp = HeadingParser()
>>> hp.feed(content)
W3C Mission
Principles
```

In [66]:
```
from html.parser import HTMLParser

class HeadingParser(HTMLParser):
    def __init__(self):
        super().__init__()
        self.in_heading = False

    def handle_starttag(self, tag, attrs):
        if tag == 'h1' or tag == 'h2' or tag == 'h3':
            self.in_heading = True
```

```
    def handle_data(self, data):
        if self.in_heading:
            print(data)

    def handle_endtag(self, tag):
        if tag == 'h1' or tag == 'h2' or tag == 'h3':
            self.in_heading = False

# Read the contents of the file
with open(r'C:\Users\chikw\Downloads\w3c (1).txt', 'r') as infile:
    content = infile.read()

# Create an instance of HeadingParser and feed it the content
hp = HeadingParser()
hp.feed(content)
```

```
W3C Mission
Principles
```

In [ ]:

Question 7

In [70]:
```
import requests
from bs4 import BeautifulSoup

def fetch_url_links(url):
    try:
        response = requests.get(url)
        soup = BeautifulSoup(response.content, 'html.parser')
        links = soup.find_all('a', href=True)
        return [link['href'] for link in links]
    except Exception as e:
        print(f"Error fetching URL {url}: {e}")
        return []

def webdir(url, depth, indent):
    if depth < 0:
        return
    print("  " * indent + url)
    links = fetch_url_links(url)
    for link in links:
        if depth > 0:
            webdir(link, depth - 1, indent + 1)

# Test the function
webdir('http://reed.cs.depaul.edu/lperkovic/csc242/test1.html', 2, 0)
```

```
http://reed.cs.depaul.edu/lperkovic/csc242/test1.html
```

In [ ]:

Question 8

In [67]:
```
Write SQL queries on the below database table that return:
a) All the temperature data.
b) All the cities, but without repetition.
c) All the records for India.
```

```
d) All the Fall records.
e) The city, country, and season for which the average rainfall is between 200 and
f) The city and country for which the average Fall temperature is above 20 degrees,
g) The total annual rainfall for Cairo.
h) The total rainfall for each season.
```

In [72]:
```
#a. SELECT Temperature FROM table_name;
#b. SELECT DISTINCT City FROM table_name;
#c. SELECT * FROM table_name WHERE Country = 'India';
#d. SELECT * FROM table_name WHERE Season = 'Fall';
#e. SELECT City, Country, Season
   #FROM table_name
    #GROUP BY City, Country, Season
    #HAVING AVG(Rainfall) BETWEEN 200 AND 400;
#f. SELECT City, Country
    #FROM table_name
    #WHERE Season = 'Fall'
    #GROUP BY City, Country
    #HAVING AVG(Temperature) > 20
    #ORDER BY AVG(Temperature) ASC;
#g. SELECT SUM(Rainfall) AS Total_Rainfall
    #FROM table_name
    #WHERE City = 'Cairo';

#h. SELECT Season, SUM(Rainfall) AS Total_Rainfall
    #FROM table_name
    #GROUP BY Season;
```

In [ ]:

QUESTION 9

In [ ]:
```
Suppose list words is defined as follows:
>>> words = ['The', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog']
Write list comprehension expressions that use list words and generate the following
a) ['THE', 'QUICK', 'BROWN', 'FOX', 'JUMPS', 'OVER', 'THE', 'LAZY', 'DOG']
b) ['the', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog']
c) [3, 5, 5, 3, 5, 4, 3, 4, 3] (the list of lengths of words in list words).
d) [['THE', 'the', 3], ['QUICK', 'quick', 5], ['BROWN', 'brown', 5], ['FOX', 'fox',
    ['JUMPS', 'jumps', 5], ['OVER', 'over', 4], ['THE', 'the', 3], ['LAZY', 'lazy',
    ['DOG', 'dog', 3]] (the list containing a list for every word of list words,
                        where each list contains the word in uppercase and lowercas
e) ['The', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog']
(the list of words in list words containing 4 or more characters.)
```

In [15]:
```
# Given list
words = ['The', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog']

# List comprehension to capitalize each word
capitalized_words = [word.upper() for word in words]

# Display the result
print(capitalized_words)
print("\n")
lowercase_words = [word.lower() for word in words]
print(lowercase_words)
print("\n")
word_lengths = [len(word) for word in words]
print(word_lengths)
print("\n")
```

```
word_info = [[word.upper(), word.lower(), len(word)] for word in words]
print(word_info)
print("\n")
long_words = [word for word in words if len(word) >= 4]
print(long_words)
```

```
['THE', 'QUICK', 'BROWN', 'FOX', 'JUMPS', 'OVER', 'THE', 'LAZY', 'DOG']


['the', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog']


[3, 5, 5, 3, 5, 4, 3, 4, 3]


[['THE', 'the', 3], ['QUICK', 'quick', 5], ['BROWN', 'brown', 5], ['FOX', 'fox',
3], ['JUMPS', 'jumps', 5], ['OVER', 'over', 4], ['THE', 'the', 3], ['LAZY', 'laz
y', 4], ['DOG', 'dog', 3]]


['quick', 'brown', 'jumps', 'over', 'lazy']
```

Question 10

In [ ]: