1. • **Differentiate between the below-mentioned operators in python.**

> **i 'and' and '&'**
>
> **ii 'or' and 'I'**

'and' vs '&':

- **'and': This is a logical operator in Python that returns True if both the operands (left and right side of 'and') are true[1].**
- **'&': This is a bitwise operator in Python that performs bit-by-bit operations.**

  Here, 14 in binary is 1110 and 10 in binary is 1010. Bitwise AND on these two will give us 1010 which is 10 in integers.

In [9]:

```
num1 = 5
num2 = 10
if num1 > 3 and num2 < 10:
    print("both are correct")
else:
    print("one is wrong")
```

```
one is wrong
```

In [10]:

```
num1 = 14
num2 = 10
print(num1 & num2)
```

```
10
```

'or' vs 'I':

- **'or': This is a logical operator in Python that returns True if at least one of the operands (left or right side of 'or') is true.**
- **'I': This is a bitwise operator in Python that performs bit-by-bit operations. It operates on the bitwise representation of integers.**

  Here, 14 in binary is 1110 and 10 in binary is 1010. Bitwise OR on these two will give us 1110 which is 14 in integers.

In [11]:

```
num1 = 5
num2 = 10
if num1 > 3 or num2 < 10:
    print("one is correct")
else:
    print("both are wrong")
```

```
one is correct
```

In [12]:

```
num1 = 14
num2 = 10
print(num1 | num2)
```

```
14
```

- **Write a python program to swap two integers without using arithmetic operators and temporary variables.**

In [13]:

```
def swap_numbers(a, b):
    a = a ^ b
    b = a ^ b
    a = a ^ b
    return a, b

# Test the function
x, y = 5, 10
x, y = swap_numbers(x, y)
print("x =", x)
print("y =", y)
```

```
x = 10
y = 5
```

- **What does the following python program print?**

```
answer = ((not (9 == 8)) and ((7 + 1) != 8)) or (6 < 4.5)
print(answer)
```

In [14]:

```
answer = ((not (9 == 8)) and ((7 + 1) != 8)) or (6 < 4.5)
print(answer)
```

```
False
```

1. • **Differentiate between "for" loop and "while" loop using appropriate examples.**

- **For Loop: A 'for' loop in Python is used to iterate over a sequence of items, such as a Python tuple, list, string, or range. The loop will execute a block of statements for each item in the sequence.**

In [15]:

```
items = ['pen', 'notebook', 'pencil', 'lunch box']
for item in items:
    print(item)
```

```
pen
notebook
pencil
lunch box
```

- **While Loop: A 'while' loop in Python is used to repeatedly execute a block of statements while a condition is true. The loop will continue to run as long as the condition remains true.**

In [16]:

```
items = ['pen', 'notebook', 'pencil', 'lunch box']
index = 0
items_len = len(items)
while index < items_len:
    print(items[index])
    index = index + 1
```

```
pen
notebook
```

```
pencil
lunch box
```

- **What is "conditional expression" in python? Write a "conditional expression" for the following python code.**

A "conditional expression" in Python, also known as the ternary operator, is a way to evaluate and return a value based on the truth or falsity of a condition. It's a shorter way to write an `if-else` statement. The syntax is: `value_if_true if condition else value_if_false`.

In [17]:

```
num = 5
print("Even Number") if num % 2 == 0 else print("Odd Number")
```

```
Odd Number
```

- **What is the output of the following python code?**

In [18]:

```
# number = 123456
# a, b = 0, 0
# while(number > 0):
#     digit = number % 10
#     if (digit % 2 == 0):
#         a += digit
#         print(a, b)
#     else:
#         b += digit
#         number //= 10

# print(a, b)

# This program has an endless loop.
```

1. - **What does a function return by default in python? Write a function which returns the area of a circle.**

- **That default return value will always be `None`**

In [19]:

```
import math

def calculate_area(radius):
    return math.pi * radius ** 2
```

- **Write a python function which takes coordinates of three points as input and returns true if points are collinear otherwise returns false.**

In [20]:

```
def are_points_collinear(point1, point2, point3):
    x1, y1 = point1
    x2, y2 = point2
    x3, y3 = point3

    slope1 = (y2 - y1) / (x2 - x1) if (x2 - x1) != 0 else float('inf')
    slope2 = (y3 - y2) / (x3 - x2) if (x3 - x2) != 0 else float('inf')

    return slope1 == slope2

# Test the function
point1 = (1, 1)
point2 = (2, 2)
point3 = (3, 3)
print(are_points_collinear(point1, point2, point3))  # Output: True

point1 = (1, 1)
point2 = (2, 2)
point3 = (3, 4)
print(are_points_collinear(point1, point2, point3))  # Output: False
```

```
True
False
```

- **What will be the output for each call made below?**
  1. **fun(2, a=3)**
  2. **fun(b=3, a=2)**
  3. **fun()**
  4. **fun(1,2)**

In [21]:

```
def fun(a=0,b=1):
    return a**b + b**a

# print(fun(2, a=3)) # TypeError
print(fun(b=3, a=2))
print(fun())
print(fun(1,2))
```

```
17
1
3
```

1. - **What is the difference between step and next command of Python debugger pdb?**

- **`step (s)` : Executes the current line and stops at the first possible occasion. If the current line is a function call, it "steps into" that function and continues execution at the first line of the function.**
- **`next (n)` : Also executes the current line, but it treats function calls as a single step and continues execution at the next line of the current function. In other words, it "steps over" function calls.**

> > **So, the main difference is that step goes into the function calls, while next goes over them, treating them as a single step.**

- **Find the logical error in the following function which returns the factorial value of an integer n.**

```
def factorial(n):
    fact=1
    for i in range(1,n):
        fact=fact*i
    return fact
```

The logical error in the function is in the range function inside the for loop. The range function should go from `1` to `n+1`, not `1` to `n`. This is because the range function in Python includes the start value but excludes the end value. So, to include `n` in the loop, you need to use `n+1` as the end value.

In [22]:

```
def factorial(n):
```

```
    fact = 1
    for i in range(1, n+1):
        fact = fact * i
    return fact
```

- **What is the output of the following python code?**

In [23]:

```
x = 10
def f():
    x = 20
    def g():
        nonlocal x
        x =30
        print(x)
    print(x)
    g()
f()
print(x)
```

```
20
30
10
```

1. - **Differentiate between split() and partition () function in string.**

- `split()` **: This function splits the string at any occurrence of the given argument. It returns a list of substrings. If the string has more than one occurrence of the separator, you will get more than two elements. For example:**

```
name = "word1 word2 word3"
print(name.split(" "))  # Output: ['word1', 'word2', 'word3']
```

If you want to split only once, you can specify the number of times to split as the second parameter:

```
name = "word1 word2 word3"
print(name.split(" ", 1))  # Output: ['word1', 'word2 word3']
```

If you are trying to split based on the whitespace characters, you don't have to pass `" "`**. You can simply do:**

```
name = "word1 word2 word3"
print(name.split())  # Output: ['word1', 'word2', 'word3']
```

- `partition()` **: This function splits the string at the first occurrence of the given argument and returns a tuple of three elements. The first element contains the part before the specified string, the second element contains the specified string, and the third element contains the part after the string[1]. For example:**

```
name = "word1 word2 word3"
first, middle, rest = name.partition(" ")
print(first, rest)  # Output: 'word1', 'word2 word3'
```

If the specified value is not found, the `partition()` **method returns a tuple containing the whole string as the first element and two empty strings as the second and third elements.**

- **Write python code to read a sentence as an input parameter and count the number of words in the sentence. For Example: if the sentence is "Programming in python" then the number of words is 3.**

In [24]:

```
def count_words(sentence):
    words = sentence.split()
    return len(words)

# Test the function
sentence = "Programming in python"
print(f"The number of words in the sentence '{sentence}' is {count_words(sentence)}")
```

```
The number of words in the sentence 'Programming in python' is 3
```

- **What is the output of the following python code?**

```
import re
string1 = "Regular expression defines a pattern"
r1 = re.findall(r"^\w+",string1)
print(r1)
print((re.split(r'\s',string1)))
```

In [25]:

```
import re
string1 = "Regular expression defines a pattern"
r1 = re.findall(r"^\w+",string1)
print(r1)
print((re.split(r'\s',string1)))
```

```
['Regular']
['Regular', 'expression', 'defines', 'a', 'pattern']
```

1. - **Write a lambda expression to compute average marks obtained by a student in three subjects, say S1, S2, and S3.**

In [26]:

```
average_marks = lambda S1, S2, S3: (S1 + S2 + S3) / 3
# Test the lambda function
S1 = 85
S2 = 90
S3 = 95
print(f"The average marks are {average_marks(S1, S2, S3)}")
```

```
The average marks are 90.0
```

- **Write a python script to find the kth smallest element of a list using Bubble Sort technique.**

In [27]:

```
def bubble_sort(lst):
    n = len(lst)
    for i in range(n):
        for j in range(0, n-i-1):
            if lst[j] > lst[j+1]:
                lst[j], lst[j+1] = lst[j+1], lst[j]
    return lst

def kth_smallest(lst, k):
    sorted_lst = bubble_sort(lst)
    return sorted_lst[k-1]

# Test the function
lst = [7, 10, 4, 3, 20, 15]
k = 3
print(f"The {k}rd smallest element is {kth_smallest(lst, k)}")
```

```
The 3rd smallest element is 7
```

- **What is the output of the following python code?**

- What is the output of the following python code?

```
n1, n2 = 6, 9
commonFactors = {i for i in range(1, min(n1+1, n2+1) #Error in this line
if(n1% i == 0 and n2 % i == 0)
print(commonFactors)
```

- What is the output of the following python code? ```python a = [1, 2, 3, 4, 5] c = [x**2 + y**2 for x, y in zip(a, a)] print(c)

In [28]:

```
a = [1, 2, 3, 4, 5]
c = [x**2 + y**2 for x, y in zip(a, a)]
print(c)
```

```
[2, 8, 18, 32, 50]
```

1. • State the difference between set and dictionary with suitable examples.

- **Set**: A set in Python is an unordered collection of unique elements. It's represented by `{}` (curly braces) and does not allow duplicate elements. For example:

```
my_set = {1, 2, 3, 4, 5}
print(my_set)  # Output: {1, 2, 3, 4, 5}
```

  To create an empty set, you can use `set()`.
- **Dictionary**: A dictionary in Python is an ordered (since Python 3.7) collection of data values, used to store data values like a map, which unlike other Data Types that hold only a single value as an element, Dictionary holds key:value pair. It's represented by `{}` (curly braces) and does not allow duplicate keys. For example:

```
my_dict = {1: "a", 2: "b", 3: "c", 4: "d", 5: "e"}
print(my_dict)  # Output: {1: 'a', 2: 'b', 3: 'c', 4: 'd', 5: 'e'}
```

  To create an empty dictionary, you can use `{}`.

- Write a function that takes a number as an input parameter and returns the corresponding text in words, for example, on input 452, the function should return 'Four Five Two'.

In [29]:

```
def number_to_words(num):
    num_to_word = {
        '0': 'Zero', '1': 'One', '2': 'Two', '3': 'Three', '4': 'Four',
        '5': 'Five', '6': 'Six', '7': 'Seven', '8': 'Eight', '9': 'Nine'
    }
    return ' '.join(num_to_word[digit] for digit in str(num))

# Test the function
num = 452
print(f"The number {num} in words is '{number_to_words(num)}'")
```

```
The number 452 in words is 'Four Five Two'
```

- What is the output of the following python code? ```python def func(): L1 = list() L2 = [ ] for i in range(0, 5): L1.append(i-1) L2.append(i+5) L1, L2 = L2, L1 print(L1) print(L2) func()

In [31]:

```
def func():
    L1 = list()
    L2 = [ ]
    for i in range(0, 5):
        L1.append(i-1)
        L2.append(i+5)
        L1, L2 = L2, L1
    print(L1)
    print(L2)
func()
```

```
[5, 0, 7, 2, 9]
[-1, 6, 1, 8, 3]
```

1. • What is recursion? What are the different parts of a recursive function?

Recursion in programming is a method where a function calls itself, usually with a different input, until an exit condition is reached. It's a way to solve complex problems by breaking them down into smaller, simpler sub-problems. The function that calls itself is known as a recursive function.

A recursive function typically consists of two main parts:

1. **Base Case**: This is the simplest scenario that does not require further recursion. It's the condition that stops the function from calling itself indefinitely. Without a proper base case, a recursive function can lead to infinite recursion[1].
2. **Recursive Case**: This is where the function calls itself with modified arguments. The recursive case should always move closer to the base case with each iteration[1].

Here's an example of a recursive function in Python that calculates the factorial of a number:

```
def factorial(n):
    # Base case: factorial of 0 is 1
    if n == 0:
        return 1
    # Recursive case
    else:
        return n * factorial(n-1)
```

In this example, the base case is when `n` is 0, and the function returns 1. The recursive case multiplies `n` with the result of the function called with `n - 1`. The process continues until the base case is reached.

- Write a python program to find the gcd of two numbers, using recursion only.

In [32]:

```
def gcd(a, b):
    if b == 0:
        return a
    else:
        return gcd(b, a % b)

# Test the function
num1 = 24
num2 = 36
print(f"The GCD of {num1} and {num2} is {gcd(num1, num2)}")
```

```
The GCD of 24 and 36 is 12
```

- Write a recursive function in python to search an element of a list using binary search technique.

In [2]:

```
def binary_search(arr, low, high, target):
    if high >= low:
        mid = (low + high) // 2

        if arr[mid] == target:
            return mid
        elif arr[mid] > target:
```

```
            return binary_search(arr, low, mid - 1, target)
        else:
            return binary_search(arr, mid + 1, high, target)
    else:
        return -1

# Test the function
arr = [2, 4, 6, 8, 10, 12, 14, 16]
target = 10
result = binary_search(arr, 0, len(arr) - 1, target)

if result != -1:
    print(f"Element {target} is present at index {result}")
else:
    print("Element is not present in the list")
```

```
Element 10 is present at index 4
```

1. • **Explain three different access modes for opening the file in python along with examples.**

Read Mode ('r'): This mode is used for reading data from a file. The file pointer is placed at the beginning of the file. If the file does not exist, a FileNotFoundError is raised. Here's an example:

```
file = open("example.txt", "r")
print(file.read())
file.close()
```

Write Mode ('w'): This mode is used for writing data to a file. If the file exists, its contents are truncated before writing. If the file does not exist, a new file is created. Here's an example:

```
file = open("example.txt", "w")
file.write("Hello, World!")
file.close()
```

Append Mode ('a'): This mode is used for appending data to a file. The file pointer is at the end of the file, so the contents are added to the end. If the file does not exist, a new file is created. Here's an example:

```
file = open("example.txt", "a")
file.write("Hello, again!")
file.close()
```

In each of these examples, the `open()` function is used to open a file with the specified mode. The `read()`, `write()`, and `close()` methods are then used to read from, write to, and close the file, respectively.

- **Write a python program to copy the contents of File 1 by reading it, into File2 by creating it and handle the error if either File1 doesn't exist.**

In [34]:

```
try:
    with open('File1.txt', 'r') as file1:
        with open('File2.txt', 'w') as file2:
            file2.write(file1.read())
    print("File copied successfully!")
except FileNotFoundError:
    print("File1 does not exist.")
```

```
File1 does not exist.
```

- **Write the type of error that will occur when following operations are performed:**
  - **m = Input('Enter value')**
  - **int('Hello')**
  - **2 + '5'**
  - **28/(14-5-4-3-2)**

In [35]:

```
m = Input('Enter value')
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[35], line 1
----> 1 m = Input('Enter value')

NameError: name 'Input' is not defined
```

In [36]:

```
int('Hello')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[36], line 1
----> 1 int('Hello')

ValueError: invalid literal for int() with base 10: 'Hello'
```

In [37]:

```
2 + '5'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[37], line 1
----> 1 2 + '5'

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

In [38]:

```
28/(14-5-4-3-2)
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
Cell In[38], line 1
----> 1 28/(14-5-4-3-2)

ZeroDivisionError: division by zero
```

1. • **Define a class Person with following data attributes: name, date of birth, and address. Moreover, you need to use init for initialization of data members. Further, explain the use of the init method. It should support following methods**

- **Define a class Rectangle. The class should contain sides: length and breadth of the rectangle as data members. It should support following methods:**
  - **init for initialization of data members.**
  - **setLength for updating the length of the rectangle.**
  - **setBreadth for updating the length of the rectangle.**
  - **area for finding the area of the rectangle.**
  - **perimeter for finding the perimeter of the rectangle.**

In [39]:

```
class Rectangle:
    def __init__(self, length, breadth):
        self.length = length
        self.breadth = breadth

    def setLength(self, length):
        self.length = length
```

```
    def setBreadth(self, breadth):
        self.breadth = breadth

    def area(self):
        return self.length * self.breadth

    def perimeter(self):
        return 2 * (self.length + self.breadth)

# Create a rectangle with length 5 and breadth 4
r = Rectangle(5, 4)

# Update the length and breadth
r.setLength(6)
r.setBreadth(5)

# Print the area and perimeter
print(f"Area: {r.area()}, Perimeter: {r.perimeter()}")
```

```
Area: 30, Perimeter: 22
```

- **What is operator overloading? Explain how the + operator is overloaded in python with an example**

Operator overloading in Python is a feature that allows the same operator to have different meanings based on the context. It means giving extended meaning beyond their predefined operational meaning. For example, the `+` operator is used to add two integers, join two strings, and merge two lists. This is achievable because the `+` operator is overloaded by the `int` and `str` classes.

To perform operator overloading, Python provides some special functions or magic functions that are automatically invoked when they are associated with that particular operator. For example, when we use the `+` operator, the magic method `__add__` is automatically invoked in which the operation for the `+` operator is defined.

Here's an example of how the `+` operator is overloaded in Python:

```
class A:
    def __init__(self, a):
        self.a = a

    # Overloading the + operator
    def __add__(self, o):
        return self.a + o.a

ob1 = A(1)
ob2 = A(2)
ob3 = A("Geeks")
ob4 = A("For")

print(ob1 + ob2)  # Output: 3
print(ob3 + ob4)  # Output: 'GeeksFor'
```