Conditions and Branching.

- `if` , `elif` , `else` keywords are used to accomplish conditions and branching in python.

```
if case1:
    perform action1

elif case2:
    perform action2

else:
    perform action3
```

- Different comparison operators shown below, can be used with if statements.
  - `Equals: a == b`
  - `Not Equals: a != b`
  - `Less than: a < b`
  - `Less than or equal to: a <= b`
  - `Greater than: a > b`
  - `Greater than or equal to: a >= b`

```
if a > b:
    print("a is greater than b")

elif a == b:
    print("a is equal to b")

else:
    print("a is less than b")
```

- In this lecture we'll cover:
  - if statement: How program enters if block.
  - Simple if else statements.
  - Nested if else statements.
  - Short Hand if....else statements / Ternary Operators
  - pass keyword

##1. if statement: How python program enters if blocks?

```
Hint: when if conditions is True
```

```
if True:
  print("if block goes here")

if False:
;
  print("if block goes here")
```

In [ ]:
```
# explain here

if True:
  print("if block goes here")


if False:
  print("If block goes here")
```

```
if block goes here
```

- Python relies on Indentation (whitespace at the beginning of a line)
  - Other programming languages often use curly-brackets for this purpose.

```
Q.Try if statement without Indentation.
```

In [ ]:
```
# write your program here.
if True:
print("if block statement")
```

```
  Cell In[2], line 3
    print("if block statement")
    ^
IndentationError: expected an indented block after 'if' statement on line
2
```

`if statement, without indentation (will raise an error)`

##2. Simple if else statements

- Generally, if statement has an optional elif, else clause.
- **syntax 1**

```
if condition:
    # block of code if condition is True

else:
    # block of code if condition is False
```

- **syntax 2**

```
if condition1:
    # code block 1

elif condition2:
    # code block 2

else:
    # code block 3
```

Q1. Input number from user, print "postive number" if number is positive else print

In [2]:
```
 ## write your program here
check= int(input("Enter a number: "))
if(check>=0):
    print("Number is positive")
else:
    print("Number is negative")
```

```
Number is negative
```

Q2. Input two numbers num1, and num2 from user,

- ***if num1 > num2: print "num1 > num2"***
- ***if num1 == num2: print "num1 = num2"***
- ***if num1 < num2: print "num1 < num2"***

In [3]:
```
## write your program here
num1=int(input("NUM1: "))
num2=int(input("NuM2: "))
if num1 > num2:
    print ("num1 > num2")
if num1 == num2:
    print ("num1 = num2")
if num1 < num2:
    print ("num1 < num2")
```

```
num1 < num2
```

##3. Nested if else statements.

- if or else statements inside if or else statements.
- **Syntax:**

```
if condition1:
  if condition:
    # code block
  else:
    # code block
```

Q. Write a python program to find max of three numbers.

```
Given three numbers num1, num2, num3

if num1 > num2:
  if num1 > num3:
    print(f"max number: {num1}")
  else:
    print(f"max number: {num3}")


else:
  if num2 > num3:
    print(f"max number: {num2}")
  else:
    print(f"max number: {num3}")
```

In [4]:
```
## write your program here
num1=int(input("NUM1: "))
num2=int(input("NuM2: "))
num3=int(input("NuM3: "))

if num1> num2:
    if num1> num3:
        print(f"max number: {num1}")
    else:
        print(f"max number: {num3}")
else:
    if num2> num3:
        print(f"max number: {num2}")
    else:
        print(f"max number: {num3}")
```

```
max number: 6
```

##4. Short Hand if...else statements / Ternary Operators

- It is way of writing if else statement in a single line of code.
- Also, called Ternary operators.
- **Syntax (simple if else)**

```
<if_block> if <condition>  else  <else_block>
```

- **Syntax (nested simple if else)**

`(<code_block> if <condition> else <code_block>)` **if** < if_condition > **else** `(<code_block> if <condition> else <code_block>)`

Q1. Write a python program to print max of 2 numbers using ternary operator.

```
Input: a = 20, b = 30
Output: Max number is 30
```

In [5]:
```
## write your Ternary program here
a=20
b=30
print(f"Max number is {a}") if a>b else print(f"Max number is {b}")
```

```
Max number is 30
```

Q2. Write a python program to print max of 3 numbers using ternary operator.

In [ ]:
```
## write your program here
# max between three numbers
```

5. Pass Keyword

- `pass` keyword acts as a placeholder for code to be added in if..else block.
- **Use case**
  - consider case, when you want to determine your python code structure in advance before adding actual code.
- **Syntax:**

```
if <condition>:
  pass
else <condition>:
  pass
```

In [6]:
```
## write your program using "pass" keyword
if True:
  pass

else:
  pass
```

Task 01: Grading System Exercise

Objective

Create a simple grading system where a student's score is entered, and the program determines the corresponding grade.

Requirements

1. Get the student's score as input.
2. Use the following grading scale:
   - 90-100: A
   - 80-89: B
   - 70-79: C
   - 60-69: D
   - Below 60: F

In [7]:
```
marks=int(input("Enter the marks: "))

def grader(marks):
    if marks<0 or marks>100:
        print("Enter valid marks")
    else:
        if marks<=100 and marks>=90:
            return "A"
        elif marks<90 and marks>=80:
            return "B"
        elif marks<80 and marks>=70:
            return "C"
        elif marks<70 and marks>=60:
            return "D"
        elif marks<60:
            return 'F'
grader(marks)
```

Out[7]:
```
'C'
```

Task 02: Ticket Price Calculator Exercise

Objective

Create a program that calculates the ticket price for a movie based on the age and whether the customer is a student.

Requirements

1. Get the user's age and whether they are a student (True or False) as input.
2. Define the ticket prices:
   - Children (age 0-12): $10
   - Teenagers (age 13-17): $15
   - Adults (age 18 and above): $20
   - Students (regardless of age): $18 (discounted price)
3. Calculate and print the ticket price based on the user's age and student status.
4. Handle cases where the entered age is not a valid numeric value.
5. Provide a message for cases where the age is negative or non-integer.

In [8]:
```
age=int(input("Enter your age "))
student = input("Are you a student [Y/N]?")
if student.lower()=='y':
    student =True
elif student.lower()=='n':
    student = False

def ticket_price(age, student):
    if(student):
        return "$18"
    else:
        if(age>=0 and age<=12):
            return "$10"
        elif(age>12 and age<=17):
            return "$15"
        elif age>=18:
            return "$20"
ticket_price(age, student)
```

Out[8]:
```
'$15'
```

Task 03: Word Frequency Counter

Objective

Create a program that analyzes a given text and counts the frequency of each unique word.

Requirements

1. Ask the user to input a paragraph or sentence.
2. Tokenize the input into words (ignoring punctuation and case sensitivity).
3. Count the frequency of each unique word.
4. Display the word frequencies in alphabetical order.
5. Handle cases where the input is empty.

In [12]:
```python

paragraph= input("Enter the paragraph ")

paragraph=paragraph.lower()
paragraph=paragraph.split()
frequency_table=dict()
def frequency(paragraph):
    for item in paragraph:
        if item in frequency_table.keys():
            frequency_table[item]+=1
        else:
            frequency_table[item]=1
    for i in sorted(frequency_table):
        print(i, frequency_table[i])
frequency(paragraph)
```

```
are 1
hello 2
how 2
today 1
we 1
```

Task 04: Magic Square Validator

Objective

Create a program that checks if a given 3x3 matrix forms a magic square.

Requirements

1. A magic square is a square matrix where the sums of the numbers in each row, column, and both main diagonals are the same.
2. Ask the user to input a 3x3 matrix (nine integer values).
3. Check and print whether the given matrix forms a magic square.
4. Handle cases where the input matrix is not of size 3x3 or contains non-integer values.

In [22]:
```
row1=input("Enter the first row")
row1=row1.split()
row1 = [eval(i) for i in row1]
print(row1)
row2=input("Enter the second row")
row2=row2.split()
row2 = [eval(i) for i in row2]
row3=input("Enter the third row")
row3=row3.split()
row3 = [eval(i) for i in row3]

matrix=[row1, row2, row3]
print(matrix)
def is_Magic_Square(matrix):
    x=matrix[0][0]+matrix[0][1]+matrix[0][2]
    print(x)
    if(matrix[1][0]+matrix[1][1]+matrix[1][2]==x) and (matrix[2][0]+matrix
        return True
    else:
        return False
print(is_Magic_Square(matrix))
```

```
[2, 7, 6]
[[2, 7, 6], [9, 5, 1], [4, 3, 8]]
15
True
```

Task 05: Palindromic Anagram Checker

Objective

Create a program that checks if a given string can be rearranged to form a palindromic string.

Requirements

1. A palindrome is a word, phrase, number, or other sequences of characters that reads the same forward and backward.
2. Ask the user to input a string.
3. Check and print whether the given string can be rearranged to form a palindrome.
4. Ignore spaces and consider the characters in a case-insensitive manner.
5. Handle cases where the input is empty or contains non-alphabetic characters.

In [28]:
```python
text=(input("Enter the text: "))
text = ''.join(list(map(lambda x: x.strip(), text.split())))
text=list(text)

frequency_table=dict()
def palindrome_checker(paragraph):
    odd_count=0
    for item in paragraph:
        if item in frequency_table.keys():
            frequency_table[item]+=1
        else:
            frequency_table[item]=1
    for value in frequency_table.values():
        if value%2!=0:
            odd_count=odd_count+1
            if(odd_count>1):
                return "Palindrome not possible."
    return "Palindrome is possible."
palindrome_checker(text)
```

Out[28]:
```
'Palindrome is possible.'
```

Congratulations, you have completed your hands-on lab in Python Conditions and Branching.

Python Loops

- Python Loops are used for executing block of code repeatedly until some condition is satisfied.
- Python loops can be categorized into 2 types:

  1. `while loop`
  2. `for loop`

- In this lecture we will cover:

  1. `while loop`
     - syntax of while loop
     - break statement
     - continue statement
     - Infinite Loop
  2. `for loop`
     - syntax of for loop
     - loop over a sequence (string, list, tuple, set, dictionary)
     - loop using range() function
     - simple vs nested for loops

1. While Loop

- With the `while` loop we can execute set of statements as long as a condition is true.
- **Syntax (simple while loop)**

```
while <condition>:
  #while block code logic
```

- **syntax (nested while loop)**

```
# outer while loop
while <condition>:

  # inner while loop
  while <condition>:
    # code logic
```

we will only look into simple while loop

```
Q. Write a python program to print "Hello world" 10 times using while
loop.
```

In [1]:
```
# write your program here
i=0
while i<10 :
    print("Hello World")
    i+=1
```

```
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
```

1.2. Break Statement

- `break` keyword is used to break and exit from the loop.

```
Q. Given list = ['mango', 'banana', 'apple', 'kiwi', 'orange']. pr
int every items, if item = apple, then exit from loop.

Output:
  - mango
  - banana
```

In [2]:
```
# write your program here
list1=['mango','banana', 'apple', 'kiwi', 'orange']
for i in list1:
    if i== 'apple':
        break
    print(i)
```

```
mango
banana
```

1.3. Continue Statement

- `continue` keyword is used to stop current iteration and continue with next iteration.

```
Q. Given list = ['mango', 'banana', 'apple', 'kiwi', 'orange'].
 **print** every items, if item = apple, simply continue with next
item without printing.

Output:
  - mango
  - banana
  - kiwi
  - orange
```

In [3]:
```
# write your program here
for i in list1:
    if i== 'apple':
        continue
    print(i)
```

```
mango
banana
kiwi
orange
```

1. 4 Infinite Loop

Q. How can I execute block of code infinite times?

`Warning: May crash your system`

In [ ]:
```
## write your program here

while True:
    print('infine loop')
```

2. for loop

- A for loop is used for iterating over a sequence (string, list, tuple, set, dictionary).
- A for loop is used using `for` keyword in python.
- **Syntax (simple for loop)**

```
for <item> in <iterable>:
  # for code block
```

- **Syntax (nested for loop)**

```
for <outer_item> in <outer_iterable>:
    # code to be executed for each outer_item

    for <inner_item> in <inner_iterable>:
        # code to be executed for each inner_item
```

2.2 Loop Over a Sequence (String, List, Tuple, Set, Dictionary)

- We have seen in the respective chapter, how we can loop through different data types.

Q.1. Given List of numbers, [10, 20, 5, 7, 2, 9, 13, 100]. Print items that are even.
`Hint: even numbers has reminder = 0`

In [4]:
```
# write your program here
num=[10,20,5,7,2,9,13,100]
for i in num:
    if not(i&1):
        print(i)
```

```
10
20
2
100
```

`Similar loop concepts will be applicable to String, Tuple, Set, Dictionary.`

2.3 Loop using range() function

- To loop through a set of code specified number of times, we can use the `range()` function.
- The `range()` function returns a sequence of numbers, starting from 0 by default. and increments by 1 (by default), and ends at a specified number.
- **Syntax:**

Q. Write a python program to print following patterns using range() function.

```
*
**
***
****
*****
```

In [11]:
```
## write your program here
for i in range(1,6):
    for j in range(i):
        print("*",end='')
    print()
```

```
*
**
***
****
*****
```

2.4 Simple vs Nested for Loops

Q.1. Write a program that takes a string as input and counts the number of vowels (a, e, i, o, u) using a for loop.

Q.2 Write a program that generates all possible combinations of three numbers from 1 to 5 using nested for loops and prints them.

In [13]:
```
## write your program for Q.1
count=0
string=list(input("Enter a string"))
for i in string:
    if i.lower() in ['a','e','i','o','u']:
        count+=1
print(count)
```

```
6
```

In [14]:
```
# write your program for Q.2
start_index = 1
end_index = 6

for num1 in range(start_index, end_index):
    for num2 in range(start_index, end_index):
        for num3 in range(start_index, end_index):
            print(num1, num2, num3)
```

```
1 1 1
1 1 2
1 1 3
1 1 4
1 1 5
1 2 1
1 2 2
1 2 3
1 2 4
1 2 5
1 3 1
1 3 2
1 3 3
1 3 4
1 3 5
1 4 1
1 4 2
1 4 3
1 4 4
```

Task 01: Word Pyramid Generator

Task

Create a program that generates a word pyramid pattern based on user input.

Objective

The objective is to generate and print a pyramid pattern using the letters of the word provided by the user. Each level of the pyramid should display the letters of the word up to that level, and the word should be centered on each level of the pyramid.

Requirements

1. Ask the user to input a word.

2. Generate and print a pyramid pattern using the letters of the word.
3. Each level of the pyramid should display the letters of the word up to that level.
4. The word should be centered on each level of the pyramid.

Additional Challenges

1. Implement a function to validate the input and ensure it's a valid word.
2. Allow the user to choose the direction of the pyramid (upwards or downwards).
3. Enhance the program to handle phrases or sentences instead of single words.

Expected Output: if word level is up:

```
              S
             S u
            S u n
           S u n i
          S u n i l
```

if word level is Down:

```
              S u n i l
               S u n i
                S u n
                 S u
                  S
```

In [33]:
```
user_input=input("Enter a word")
direction=input("Enter direction ['up'/'down']:")
if(direction.lower()=='up'):
    for i in range(len(user_input)):
        for j in range(len(user_input)-i-1):
            print(" ",end='')
        for j in range(i+1):
            print (user_input[j]+" ",end='')
        print()
if(direction.lower()=='down'):
    for i in range(len(user_input)):
        for j in range(i+1):
            print(" ",end='')
        for j in range(len(user_input)-i):
            print (user_input[j]+" ",end='')
        print()
```

```
 s u n i l
  s u n i
   s u n
    s u
     s
```

Task 02: List Manipulation - Odd-Even Sorter

Objective

Create a program that takes a list of numbers from the user, sorts them into two separate lists (one for odd numbers and one for even numbers), and displays the sorted lists.

Requirements

1. Ask the user to input a list of numbers (comma-separated).
2. Sort the numbers into two lists: one for odd numbers and one for even numbers.
3. Display both lists.

Additional Challenges

1. Allow the user to input any type of values (not just numbers) and handle different data types.
2. Enhance the program to display the sorted lists in ascending or descending order.

In [37]:
```
even=[]
odd=[]
user_input=input("Enter the list of number seperated with commas")
order=input("Enter the order [ascending/descending]?")
user_input= list(map(lambda x: x.strip(), user_input.split(',')))
user_input=[eval(i) for i in user_input]
print (user_input)
for i in user_input:
    if i&1:
        odd.append(i)
    else:
        even.append(i)
if order.lower()=="ascending":
    print(sorted(even))
    print(sorted(odd))
if order.lower()=='descending':
    print(sorted(even,reverse=True))
    print(sorted(odd,reverse=True))
```

```
[2, 3, 4, 5, 6]
[2, 4, 6]
[3, 5]
```

Task 03: Prime Factorization

Objective

Create a program that takes an integer input from the user and prints its prime factorization.

Requirements

1. Ask the user to input a positive integer.
2. Compute and print the prime factorization of the input integer.

Additional Challenges

1. Implement error handling to ensure the user inputs a valid positive integer.
2. Allow the program to handle edge cases, such as the input being 1 or a prime number.
3. Enhance the program to handle multiple integer inputs in a loop until the user chooses to exit.

In [51]:
```
input_num=int(input("Enter a number"))
factors=[]
def isPrime(i):
    for j in factors:
        if(i%j==0):
            return
    factors.append(i)

for i in range(2,int((input_num)**1/2)):
    if(input_num%i==0):
        isPrime(i)
print(factors)
```

```
[2, 13]
```

Task 04: Number Guessing Game

Objective

Create a simple number guessing game where the program generates a random number, and the user has to guess it.

Requirements

1. Generate a random number between a specified range.
2. Ask the user to guess the number.
3. Provide feedback on whether the guess is too high, too low, or correct.
4. Allow the user to continue guessing until they guess the correct number.
5. Display the number of attempts it took to guess correctly.

Additional Challenges

1. Implement error handling to ensure the user inputs a valid number.
2. Allow the user to choose the range of numbers for the guessing game.
3. Enhance the program to provide hints or clues based on the user's previous guesses.

In [52]:
```
import random

start=int(input("Start range: "))
end=int(input("End: "))
target=random.randint(start,end)
def guess_number(target,start,end):
    while(start<end):
        guess=int(input(f"Guess the number between {start} and {end}"))
        if guess==target:
            return "Correct"
        if guess> target:
            print("guess is higher")
        if guess<target:
            print("guess is low")
guess_number(target,start,end)
```

```
guess is low
guess is low
guess is low
```

Out[52]:
```
'Correct'
```

Congratulations, you have completed your hands-on lab in Python Loops:.

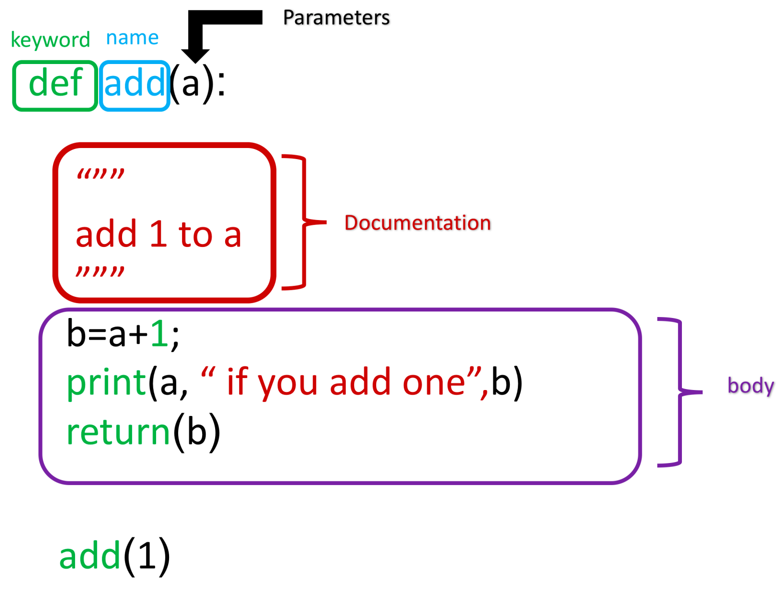
In [ ]:
```
```

Python Functions

- A function is a reusable block of code which only runs when it is called.
- You can pass data, known as parameters, into a function.
- A function can return data as result.
- Function allow us to not have to repeatedly write the same code again and again.
  - Recall `len()` function that we used to compute length of iterables such as String, List, Tuple, Set, Dictionary.
- Generally, we create a function and write a block of code that will be used repeatedly in future.
- **When to use functions?**
  - Use functions when you plan on using a block of code multiple times.
- **Types:**
  - Pre-defined functions
  - User defined functions
- In this lecture we'll cover:
  - `Anatomy of function`
  - `Create function without parameters and without return statement`
  - `Create function with parameters and without return statement`
  - `Create function without parameters and with return statement`
  - `Create function with parameters and with return statement`
  - `local variable vs global variable`
  - `Arguments vs Parameters`
  - `Arbitrary Positional Arguments, *args`
  - `Keyword Arguments`
  - `Arbitrary Keyword Arguments, **kwargs`
  - `Default Parameter Value`
  - `pass keyword`
  - `Recall Pre-defined functions`

1. Anatomy of function

- `def` keyword is used for creating a uder-defined function in python.
- **syntax:**



- function begins with `def` keyword followed by function `name` and parenthesis `()`
- Function has body that starts with colon (`:`) and is indented.
- You can add documentation in a function also called as `doc string`

2. Create a function without parameters and without return statement.

- Function without return statement are called Void function.

- **Syntax:**

```
def function_name():
  """
  doc string goes here (optional)
  """
  # code logic goes
```

Q. Write a python function that takes two integer input from user and print greater number. *(no parameter, no return)*

```
Hint: a if a > b else a
```

In [ ]:
```
## your program goes here
# function creation part

def print_greater():
    """
    print greater between 2 numbers
    """
    num1 = int(input("enter number1: "))
    num2 = int(input("enter number2: "))

    if num1 > num2:
        print("num1 is greater")

    else:
        print("num2 is greater")
```

In [ ]:
```
# call your function from here
print_greater()
```

```
num2 is greater
```

3. Create function with parameters and without return statement

---

```
def function_name(parameter):
  """
  doc string goes here (optional)
  """
  # code logic goes
```

Q. Write a python function that takes two integer input from user pass them as argument to function and print greater number. *(no return)*

```
Hint: a if a > b else b
```

In [3]:
```
## write your program here
def print_greater(num1, num2):
  """
  function that prints greater between two numbers.

  Parameters: num1 (int), num2(int)
  """

  if num1>num2:
    print("num1 is greater")

  else:
    print("num2 is greater")
```

In [4]:
```
num1 = int(input("enter number1: "))
num2 = int(input("enter number2: "))

print_greater(num1, num2)
```

```
num2 is greater
```

##4. **Create function without parameters and with return statement**

- Functions with return statement are called `Fruitful function` .

```
def function_name():
  """
  doc string goes here (optional)
  """
  # code logic goes

  # return <item>
```

Q. Write a python function that takes two integer input from user inside function and return greater number from function. *(no parameter)* `Hint: a if a > b else b`

In [5]:
```python
## write your program here
def print_greater():
  num1 = int(input("enter number1: "))
  num2 = int(input("enter number2: "))

  if num1 > num2:
    return "num1 is greater"

  else:
    return "num2 is greater"

result = print_greater()

print(result)
```

```
num2 is greater
```

5. Create function with parameters and with return statement

```
    def function_name(parameter):
      """
      doc string goes here (optional)
      """
      # code logic goes

      # return <item>
```

Q. Write a python function that takes two integer as a argument and return greater number from function.

```
Hint: a if a > b else b
```

In [6]:
```python
## write your program here
def print_greater(num1, num2):
  if num1 > num2:
    return "num1 is greater"
  else:
    return "num2 is greater"
```

In [7]:
```
num1 = int(input("enter number1: "))
num2 = int(input("enter number2: "))

result = print_greater(num1, num2)

print(result)
```

```
num2 is greater
```

In [7]:
```
def test_program(*args, **kwargs):
    print(args)
    print(kwargs)


if __name__=="__main__":
    name= list(["xyz","abs"])
    test_program( *name,age="23")
```

```
('xyz', 'abs')
{'age': '23'}
```

##6. **Local Variable vs Global Variable**

	Local Variables	Global Variables
Scope	Limited to the function or block	Accessible from any part of the program
Declaration	Defined within a function or block	Defined outside any function or block
Accessibility	Only within the function or block	Anywhere in the program
Lifetime	Created when function/block is executed	Exist until the program terminates
Naming Conflicts	Avoids naming conflicts within functions	Prone to potential naming conflicts
Memory Allocation	Allocated when function/block is executed	Allocated when the program starts
Initialization	Initialized when assigned a value	Initialized when assigned a value
Modifiability	Can be modified within the function/block	Can be modified from any part of the program
Accessing and Modifying	Directly using the variable name	Using the `global` keyword

```
6.1. Local Variables
```

In [8]:
```
## write your program here

def my_function():
    x = 10  # Local variable
    print(x)

my_function()  # Output: 10
print(x)      # Error: NameError: name 'x' is not defined
```

```
10

---------------------------------------------------------------------------
--
NameError                                 Traceback (most recent call las
t)
Cell In[8], line 8
      5     print(x)
      7 my_function()  # Output: 10
----> 8 print(x)      # Error: NameError: name 'x' is not defined

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

6.2 Global Variables

In [9]:
```
## write your program here

x = 10  # global variable

def my_function():
    global x # global keyword to update global variable
    x += 5
    print("I am from function: ", x)

my_function()  # Output: 15

print("I am from outside of function: ", x)      # Output: 15
```

```
I am from function:  15
I am from outside of function:  15
```

It's generally considered good practice to limit the use of global variables and use local variables instead

In [ ]:

##7. **Arguments vs Parameters**

Concept	Parameters	Arguments
Definition	Variables listed in the function's header	Actual values passed to a function
Purpose	Define input placeholders in a function	Provide data for the function to operate on
Scope	Local to the function	Specific to each function call

Concept	Parameters	Arguments
Usage	Perform operations or calculations	Act as input for the function's computations

```
In [ ]:
```

##8. **Arbitrary Positional Arguments, *args**

- Positional arguments are passed to a function based on their position or order in the function call.
- The arguments we passed to a function till now are all positional arguments.
- The `*args` syntax allows a function to accept any number of positional arguments.
- The variable type of `args` will be of Tuple.

```
def my_function(*args):
    for arg in args:
        print(arg)

my_function(1, 2, 3)        # Output: 1 2 3
my_function('Hello', 'Hi')  # Output: Hello Hi
```

`Q. Write a python function that return multiplication of all the numbers passed by user as a argument.`

`Note: User can pass unlimited numbers to a function at the time of function call`

```
In [11]: ## write your program here

def dynamic_multiplication(*args):
  mult = 1

  for item in args:
    mult = mult * item

  return mult

result = dynamic_multiplication(1, 1, 2, 3, 4)
print(result)
```

```
24
```

9. Keyword Arguments

- Keyword arguments in Python allow you to pass arguments to a function using their corresponding parameter names.
- Instead of relying on the order of the arguments, you specify the argument name along with its value when calling the function.
- **Key Benefit**
  - `Clarity:`
    - makes function call more readable and self-explnatory.
  - `Order Independence:`

    - all you to provide the values in any order you prefer, as long as the parameter names are correctly matched.
- **Example:**

```
def greet(name, age):
  """
  Function that greets to person.
  """
  print(f"Hi {name}. Your age is {age}")


greet(name='John', age=30) # ouput: Hi John. Your age is 30

greet(age=10, name='Bob') # output: Hi Bob. Your age is 10
```

In [12]:
```
## write your program here

def greet(name, age):
  """
  Function that greets to person.
  """
  print(f"Hi {name}. Your age is {age}")


greet(name='John', age=30) # ouput: Hi John. Your age is 30

greet(age=10, name='Bob') # output: Hi Bob. Your age is 10
```

```
Hi John. Your age is 30
Hi Bob. Your age is 10
```

In [ ]:

##10. **Arbitrary Keyword Arguments, **kwargs**

- If you do not know how many keyword arguments that will be passed into your function, add two asterik: `**` .
- This way, the function will receive a dictionary of arguments, and can access items accordingly.
- **Example:**

```
def my_function(**kwargs):
    for key, value in kwargs.items():
        print(f"{key}: {value}")

greet(name='John', age=30) # ouput: Hi John. Your age is 30

greet(age=10, name='Bob') # output: Hi Bob. Your age is 10
```

In [13]:
```
## write your program here
def my_function(**kwargs):
    for key, value in kwargs.items():
        print(f"{key}: {value}")

greet(name='John', age=30) # ouput: Hi John. Your age is 30

greet(age=10, name='Bob') # output: Hi Bob. Your age is 10
```

```
Hi John. Your age is 30
Hi Bob. Your age is 10
```

##11. **Default Parameter Value**

`What if you do not want to pass arguments at the time of function call but function is expecting some parameters?`

`Ans:` *You can use concept of Default parameter value.`*

```
def print_message(msg = 'hello_world'):
  print(msg)

print_message('How are you!!') # output: How are you!!
print_message(msg='What is your name?') # output: What is your nam
e?
print_message() # output: hello_world
```

In [14]:
```
# write your program here
def print_message(msg = 'hello_world'):
  print(msg)

print_message('How are you!!') # output: How are you!!
print_message(msg='What is your name?') # output: What is your name?
print_message() # output: hello_world
```

```
How are you!!
What is your name?
hello_world
```

12. Pass Keyword

- `functions` definition cannot be empty.
- `Q. What if you only want to define a function and write code logic later?`
- In such scenario we can use `pass` keyword, which simply acts as a placeholder for code logic to be added later.

```
def function1():
  pass
```

Now, we know user defined functions, How functions are created, Let's Revisit some pre-defined functions.

##13. **Recall Pre-defined functions**

- Pre-defined functions, also know as built-in functions, are functions that are included in the Python programming language by default.

Function	Description
`print()`	Outputs messages or values to the console.
`len()`	Returns the length of a sequence (e.g., string, list, tuple).
`type()`	Returns the type of an object.
`input()`	Reads user input from the console.
`range()`	Generates a sequence of numbers within a specified range.
`sum()`	Calculates the sum of all the values in an iterable.
`max()`	Returns the maximum value in an iterable.
`min()`	Returns the minimum value in an iterable.
`abs()`	Returns the absolute value of a number.
`round()`	Rounds a number to a specified number of decimal places.
`str()`	Converts an object to a string representation.
`int()`	Converts a string or a number to an integer.
`float()`	Converts a string or a number to a floating-point number.
`list()`	Converts an iterable to a list.
`dict()`	Creates a new dictionary object.
`sorted()`	Returns a new sorted list from the items in an iterable.
`zip()`	Creates an iterator that aggregates elements from multiple iterables.
`enumerate()`	Returns an iterator of tuples containing indices and values from an iterable.
`help()`	Displays information and documentation about Python objects and modules.
`dir()`	Returns a list of names in the current local scope or a given object.

13.1 Math Module

- The `math` module in Python provides a set of mathematical functions and constants for various mathematical operations.
- It is part of the Python Standard Library, so no additional
- However you need to import math module, to access functions and constants.

Function	Description
`math.ceil(x)`	Returns the smallest integer greater than or equal to x.
`math.floor(x)`	Returns the largest integer less than or equal to x.
`math.sqrt(x)`	Returns the square root of x.

Function	Description
`math.pow(x, y)`	Returns x raised to the power of y.
`math.exp(x)`	Returns e raised to the power of x.
`math.log(x)`	Returns the natural logarithm (base e) of x.
`math.log10(x)`	Returns the base-10 logarithm of x.
`math.sin(x)`	Returns the sine of x (in radians).
`math.cos(x)`	Returns the cosine of x (in radians).
`math.tan(x)`	Returns the tangent of x (in radians).
`math.radians(x)`	Converts x from degrees to radians.
`math.degrees(x)`	Converts x from radians to degrees.
`math.pi`	Represents the mathematical constant π (pi).
`math.e`	Represents the mathematical constant e (Euler's number).
`math.factorial(x)`	Returns the factorial of x.

In [15]:
```
# import module
import math

math
```
Out[15]:
```
<module 'math' (built-in)>
```

In [16]:
```
# floor and ceil
num = 3.9

print("Floor: ", math.floor(num))
print("Ceil: ", math.ceil(num))
```
```
3
4
```

In [17]:
```
# factorial of number
math.factorial(5)
```
Out[17]:
```
120
```

In [ ]:
```

```

Recursion

- Recursion is process, where a function calls itself.

```
factorial(7) = 7*6*5*4*3*2*1
factorial(6) = 6*5*4*3*2*1
factorial(5) = 5*4*3*2*1
factorial(4) = 4*3*2*1
::::::::::::
factorial(0) = 1

factorial(n) = n * factorial(n-1)
```

In [18]:

```
# write your program

def factorial(n):
  if (n==0 or n==1):
    return 1

  else:
    return n * factorial(n-1) # calling same function, called Recursion

print(factorial(3))
print(factorial(4))
print(factorial(5))
```

```
6
24
120
```

Assignments

Q.1 Write a python function to multiply all the numbers in a list

Q.2 Write a python function to reverse a string

Q.3 Write a python function to find factorial of a given non negativenumber

Q.4 Write a python function to reverse a string

Q.5 Write a python function that accepts a string and calculate the number of upper case letters and lower case letters

Q.6 Write a python function that takes a list and returns a new list with unique elements of the first list

**Q.7 Write a Python function that takes a number as a parameter and check the
number is prime or not.**

Q.8 Write a python function to print the even numbers from a given list.

Congratulations, you have completed your hands-on lab in Python Functions.

In [ ]:

Advanced Inheritance and Abstract Classes

Objective

Create a set of classes representing different animals, introducing multiple levels of inheritance and abstract classes.

Requirements

1. Create an abstract class called `Animal` with abstract methods:
    - `speak` : Abstract method representing the sound the animal makes.
    - `move` : Abstract method representing how the animal moves.
    - `eat` : Abstract method representing what the animal eats.
2. Implement three concrete classes: `Mammal` , `Bird` , and `Fish` , inheriting from the `Animal` class. Implement the abstract methods accordingly.
3. Create concrete classes for specific animals within each category:
    - For `Mammal` : Implement classes like `Dog` and `Cat` .
    - For `Bird` : Implement classes like `Eagle` and `Penguin` .
    - For `Fish` : Implement classes like `Salmon` and `Goldfish` .
4. Add unique methods for each specific animal:
    - For example, `bark` for `Dog` , `fly` for `Eagle` , `swim` for `Salmon` .
5. Demonstrate the usage of these classes by creating instances and calling various methods.

Advanced Inheritance with Multiple Levels

Objective

Create a set of classes representing different types of vehicles, introducing multiple levels of inheritance, and demonstrating the use of `super().__init__` .

Requirements

1. Create a base class called `Vehicle` with the following attributes:
    - `make` : Make of the vehicle (e.g., Ford, Honda).
    - `model` : Model of the vehicle (e.g., Civic, F-150).
    - `year` : Year of manufacture.
    - `fuel_type` : Type of fuel the vehicle uses (e.g., Gasoline, Electric).
2. Create two subclasses: `Car` and `Truck` , inheriting from the `Vehicle` class. Implement the `__init__` method using `super().__init__` to initialize attributes from the parent class.
3. Create a subclass of `Car` called `ElectricCar` . Add an additional attribute:

   - `battery_capacity` : Capacity of the electric car's battery in kWh.
4. Create a subclass of `Truck` called `HybridTruck`. Add an additional attribute:
   - `electric_motor_power` : Power of the electric motor in the hybrid truck.
5. Demonstrate the usage of these classes by creating instances and displaying information about the vehicles.

In [ ]:

```
```

Type *Markdown* and LaTeX: α^2