

**Programming for AI**

***Assignment # 01***

**Submitted by:** Muhammad Haseeb

**Registration No:** B23F0083AI116

**Submitted to:** Ms. Maria Gul

**Class/Section:** AI-Red

**Department:** IT & CS

**Semester:** 03

*Date: September 28, 2024*



***Task 01:***

***You are working in a SmartFormat AI company as a Python programmer. You***

***have been assigned a project in which you will create a chatbot. The conditions***

***for creating a chatbot are:***

***1. Needs to remove the unnecessary blank spaces.***

***2. Consistently needs formatting text.***

***3. Needs to remove the unwanted characters.***

***4. The chatbot needs the user’s name, and a greeting message and also***

***removes the duplicated words and replaces these words with proper***

***symbols or spaces.***

***5. The greeting message must be in upper case letter.***

***6. If the user inserts any keyword from his/her entered statement, then the***

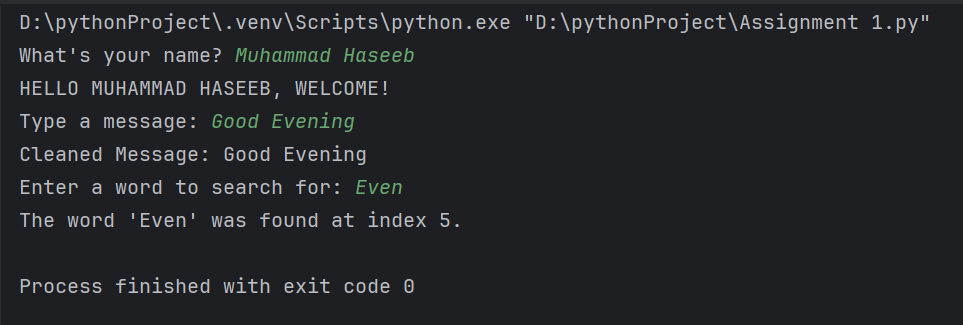
***chatbot must display its index.***

***Code:***

*# Function to clean up user input  
def clean\_text(text):  
 # Remove extra spaces  
 text = ' '.join(text.split())  
  
 # Remove unwanted characters  
 for char in [',', '.', '!', '?', '@', '#', '$', '%', '^', '&', '\*', '(', ')']:  
 text = text.replace(char, '')  
  
 # Remove duplicate words  
 words = text.split()  
 unique\_words = []  
 for word in words:  
 if word not in unique\_words:  
 unique\_words.append(word)  
  
 # Return cleaned text  
 return ' '.join(unique\_words)  
  
  
# Function to greet the user in uppercase  
def greet(name):  
 greeting = f"HELLO {name.upper()}, WELCOME!"  
 return greeting  
  
  
# Function to find keyword in the text and return its index  
def find\_word(text, keyword):  
 if keyword in text:  
 return text.index(keyword)  
 return -1 # If keyword is not found  
  
  
# Main chatbot function  
def chatbot():  
 # Get user's name  
 user\_name = input("What's your name? ")  
  
 # Display greeting in uppercase  
 print(greet(user\_name))  
  
 # Get user's message  
 user\_message = input("Type a message: ")  
  
 # Clean and format the message  
 cleaned\_message = clean\_text(user\_message)  
 print("Cleaned Message:", cleaned\_message)  
  
 # Ask for a keyword to find  
 search\_word = input("Enter a word to search for: ")  
  
 # Find the word's index in the message  
 word\_index = find\_word(cleaned\_message, search\_word)  
  
 if word\_index != -1:  
 print(f"The word '{search\_word}' was found at index {word\_index}.")  
 else:  
 print(f"The word '{search\_word}' was not found in the message.")  
  
  
# Run the chatbot  
chatbot()*

***Output***

The output of the above code is shown below:



***Task 02:***

***Donald is a customer who wants you to help, and his requirements are:***

***1. When Donald enters any phrase, then it must be printed as it is.***

***2. After printing as it is, Donald wants the reverse version of that phase.***

***3. The restriction from Donald's side is the program must ignore white***

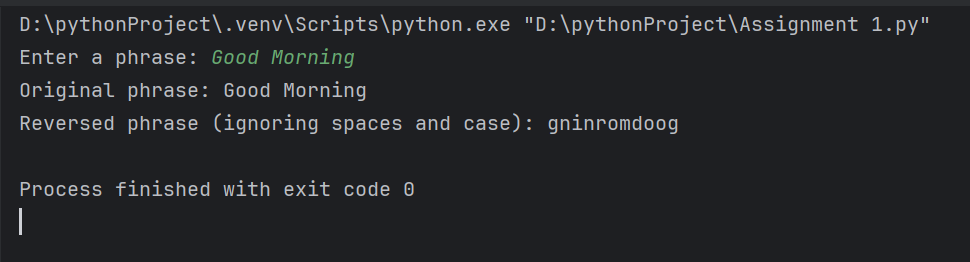
***spaces and cases.***

***Code:***

*# Function to reverse a phrase ignoring spaces and case  
def reverse\_phrase(phrase):  
 # Remove white spaces using replace() and convert to lowercase using lower()  
 cleaned\_phrase = phrase.replace(' ', '').lower()  
  
 # Reverse the cleaned phrase using slicing  
 reversed\_phrase = cleaned\_phrase[::-1]  
  
 # Return the reversed phrase  
 return reversed\_phrase  
  
  
# Main program to handle Donald's request  
def handle\_donald():  
 # Step 1: Get the phrase input from Donald  
 user\_input = input("Enter a phrase: ")  
  
 # Step 2: Print the phrase as it is  
 print("Original phrase:", user\_input)  
  
 # Step 3: Get the reversed phrase (ignoring spaces and case)  
 reversed\_output = reverse\_phrase(user\_input)  
  
 # Step 4: Print the reversed version of the phrase  
 print("Reversed phrase (ignoring spaces and case):", reversed\_output)  
  
  
# Run the program  
handle\_donald()*

***Output***

The output of the above code is shown below:



***Task 03:***

***Five friend groups went for lunch, and they ordered their favourite dish and***

***enjoyed them. Know it's time to pay the bill. To pay the bill, the friends want***

***to divide the total bill among themselves and also the services from the***

***restaurant were good, they also want to give tip to the waiter. The problem is***

***they don't know about the divided bill as well as tips. They need a program***

***where they can enter their total number of friends, and their dish's total price,***

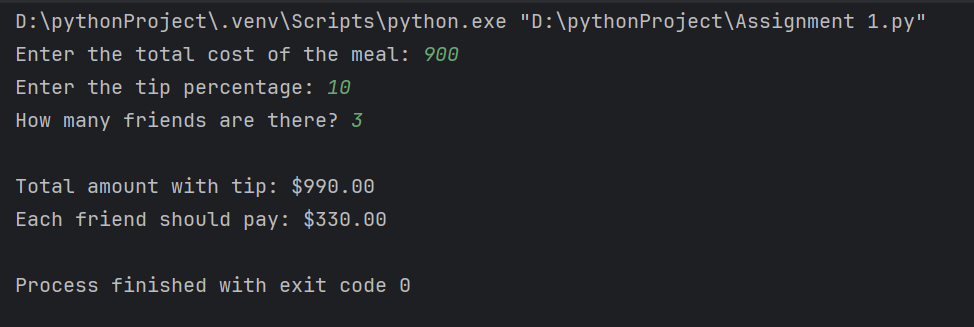
***then the total bill is divided among five friends and the tip for the waiter.***

***Code:***

*# Function to calculate the total bill with tip and split it among friends  
def split\_bill(total\_cost, tip\_percent, friends\_count):  
 # Step 1: Calculate the tip amount  
 tip = (tip\_percent / 100) \* total\_cost  
  
 # Step 2: Calculate the total amount including the tip  
 total\_amount = total\_cost + tip  
  
 # Step 3: Divide the total amount among friends  
 per\_person = total\_amount / friends\_count  
  
 # Return the total amount and amount each friend pays  
 return total\_amount, per\_person  
  
  
# Main function to get input and display the results  
def bill\_calculator():  
 # Step 1: Get the total cost of the meal  
 meal\_cost = float(input("Enter the total cost of the meal: "))  
  
 # Step 2: Get the percentage of tip  
 tip\_percent = float(input("Enter the tip percentage: "))  
  
 # Step 3: Get the number of friends  
 friends = int(input("How many friends are there? "))  
  
 # Step 4: Call the split\_bill function to calculate totals  
 total\_bill, each\_share = split\_bill(meal\_cost, tip\_percent, friends)  
  
 # Step 5: Display the total amount with tip and the per person share  
 print(f"\nTotal amount with tip: ${total\_bill:.2f}")  
 print(f"Each friend should pay: ${each\_share:.2f}")  
  
  
# Run the program  
bill\_calculator()*

***Output***

The output of the above code is shown below:



***Task 04:***

***Take phrases from a user, and calculate the total word number, sentences, and***

***characters in the phrases. Calculate the total average word length and phrase***

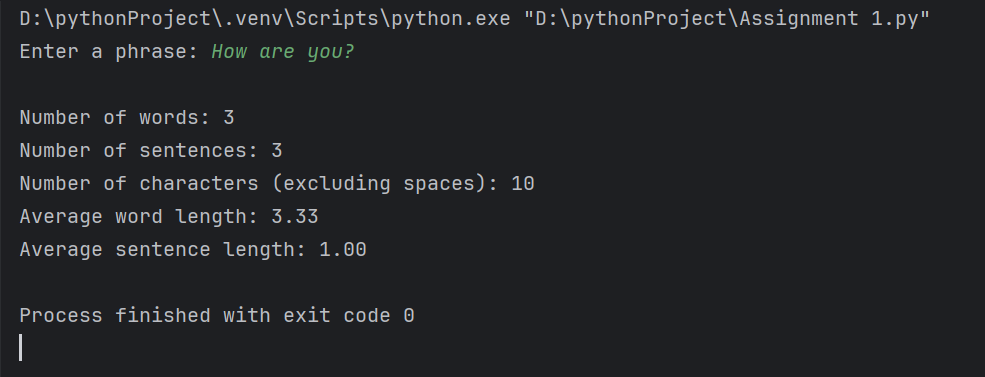
***length and print out the output.***

***Code:***

*# Function to calculate word, sentence, and character counts  
def calculate\_stats(text):  
 # Step 1: Count the number of words by splitting the text  
 word\_list = text.split() # Split by spaces  
 num\_words = len(word\_list)  
  
 # Step 2: Count the number of sentences by splitting on punctuation  
 sentence\_list = text.split('.')  
 sentence\_list += text.split('!')  
 sentence\_list += text.split('?')  
 num\_sentences = len([s for s in sentence\_list if s.strip() != '']) # Only count non-empty sentences  
  
 # Step 3: Count the number of characters (excluding spaces)  
 total\_chars = len(text.replace(' ', '')) # Remove spaces to count characters  
  
 # Step 4: Calculate the average word length  
 if num\_words > 0:  
 avg\_word\_length = total\_chars / num\_words  
 else:  
 avg\_word\_length = 0  
  
 # Step 5: Calculate the average sentence length  
 if num\_sentences > 0:  
 avg\_sentence\_length = num\_words / num\_sentences  
 else:  
 avg\_sentence\_length = 0  
  
 # Return the results  
 return num\_words, num\_sentences, total\_chars, avg\_word\_length, avg\_sentence\_length  
  
  
# Main function to handle user input and display the stats  
def text\_analyzer():  
 # Step 1: Get input text from the user  
 user\_input = input("Enter a phrase: ")  
  
 # Step 2: Call the function to calculate statistics  
 words, sentences, characters, avg\_word\_len, avg\_sentence\_len = calculate\_stats(user\_input)  
  
 # Step 3: Display the results  
 print(f"\nNumber of words: {words}")  
 print(f"Number of sentences: {sentences}")  
 print(f"Number of characters (excluding spaces): {characters}")  
 print(f"Average word length: {avg\_word\_len:.2f}")  
 print(f"Average sentence length: {avg\_sentence\_len:.2f}")  
  
  
# Run the program  
text\_analyzer()*

***Output***

The output of the above code is shown below:



***Task 05:***

***If I leave my house at 4:25 am for running and I run 1 mile at an easy pace like***

***5:15 per mile, then 3 miles at a tempo which is almost 6:15 per mile and 2***

***miles at an easy pace again, what time do I get home for breakfast and ready***

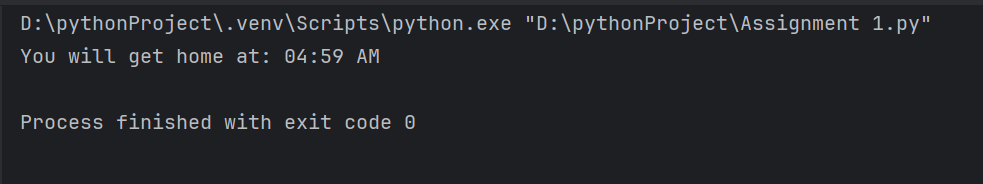
***for university?***

***Code:***

*from datetime import datetime, timedelta  
  
  
# Function to calculate the total running time  
def calculate\_running\_time():  
 # Step 1: Define the paces in minutes per mile  
 easy\_pace = 5 \* 60 + 15 # 5 minutes 15 seconds per mile in total seconds  
 tempo\_pace = 6 \* 60 + 15 # 6 minutes 15 seconds per mile in total seconds  
  
 # Step 2: Calculate the time for each segment of the run  
 time\_easy\_1\_mile = easy\_pace \* 1 # Time for 1 mile at easy pace  
 time\_tempo\_3\_miles = tempo\_pace \* 3 # Time for 3 miles at tempo pace  
 time\_easy\_2\_miles = easy\_pace \* 2 # Time for 2 miles at easy pace  
  
 # Step 3: Total running time in seconds  
 total\_running\_time = time\_easy\_1\_mile + time\_tempo\_3\_miles + time\_easy\_2\_miles  
  
 # Convert total running time to a timedelta object  
 total\_time\_delta = timedelta(seconds=total\_running\_time)  
  
 return total\_time\_delta  
  
  
# Main function to calculate the time you get home  
def calculate\_return\_time():  
 # Step 1: Define the time you leave the house (4:25 AM)  
 leave\_time = datetime.strptime("04:25 AM", "%I:%M %p")  
  
 # Step 2: Calculate total running time  
 total\_run\_time = calculate\_running\_time()  
  
 # Step 3: Calculate the time you get home by adding the running time to the leave time  
 return\_time = leave\_time + total\_run\_time  
  
 # Step 4: Print the time you will get home in a readable format  
 print("You will get home at:", return\_time.strftime("%I:%M %p"))  
  
  
# Run the program  
calculate\_return\_time()*

***Output***

The output of the above code is shown below:



***Task 06:***

***A “string” is a very valuable data type in Python. Python programming***

***language offers a rich set of string processing and manipulation operations.***

***Briefly describe the following operations on strings:***

***1. Logical operators (<, <=, >, >=, ==, !=)***

***2. Membership operators (“in” and “not in”)***

***3. Escape sequences supported by Python***

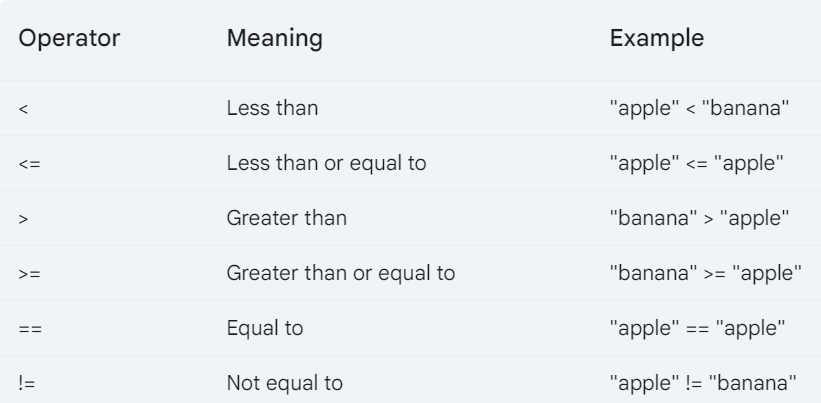
***4. Using “+” and “ \* ” operators on Strings***

***Solution:***

*String Operations in Python*

*1. Logical Operators:*

*Logical operators are used to compare strings based on their lexicographic order (alphabetical order).*

**

*2. Membership Operators:*

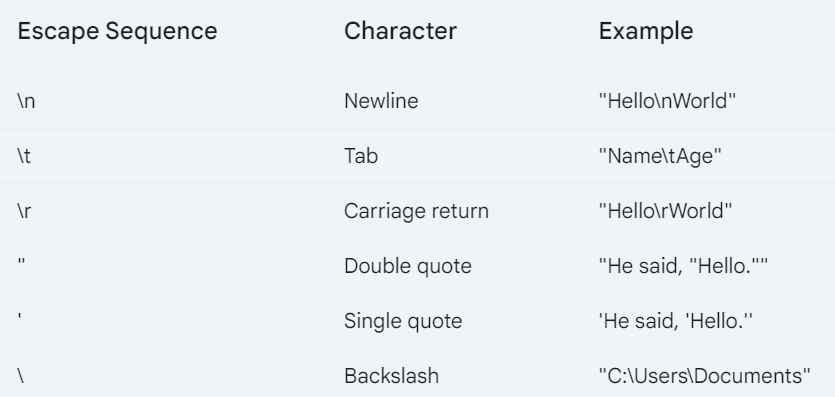
*Membership operators check if a substring exists within a string.*

*A screenshot of a phone

Description automatically generated*

*3. Escape Sequences:*

*Escape sequences are used to represent special characters within a string.*

**

*4. Using "+" and "\*" Operators:*

* *Concatenation (+): Joins two strings together.*

*string1 = "Hello" string2 = "World" result = string1 + " " + string2 print(result) # Output: Hello World*

*Replication (\*): Repeats a string multiple times.*

*string = "Python" result = string \* 3 print(result) # Output: PythonPythonPython*

*Complete Example:*

*# Logical operators string1 = "apple" string2 = "banana" if string1 < string2: print("apple comes before banana") # Membership operators text = "I like apples and oranges" if "apple" in text: print("apple is in the text") # Escape sequences print("Hello\nWorld") print("Name\tAge") # Concatenation first\_name = "John" last\_name = "Doe" full\_name = first\_name + " " + last\_name print(full\_name) # Replication greeting = "Hello" repeated\_greeting = greeting \* 3 print(repeated\_greeting)*

***Task 07:***

***Objects in Python have both attributes and methods. List down different***

***attributes and their operation that a string variable has.***

***Solution:***

*String Attributes in Python*

*String objects in Python have several attributes that provide information about the string's content and properties.*

*Common String Attributes:*

1. *len(string): Returns the length of the string (number of characters).*

*Python*

*my\_string = "Hello, world!"*

*length = len(my\_string)*

*print(length) # Output: 13*

1. *string.upper(): Converts all characters in the string to uppercase.*

*Python*

*my\_string = "hello"*

*uppercase\_string = my\_string.upper()*

*print(uppercase\_string) # Output: HELLO*

1. *string.lower(): Converts all characters in the string to lowercase.*

*Python*

*my\_string = "HELLO"*

*lowercase\_string = my\_string.lower()*

*print(lowercase\_string) # Output: hello*

1. *string.capitalize(): Capitalizes the first character of the string.*

*Python*

*my\_string = "hello, world"*

*capitalized\_string = my\_string.capitalize()*

*print(capitalized\_string) # Output: Hello, world*

1. *string.title(): Capitalizes the first character of each word in the string.*

*Python*

*my\_string = "hello, world"*

*title\_string = my\_string.title()*

*print(title\_string) # Output: Hello, World*

1. *string.swapcase(): Swaps the case of all characters in the string (uppercase to lowercase and vice versa).*

*Python*

*my\_string = "Hello, World!"*

*swapped\_string = my\_string.swapcase()*

*print(swapped\_string) # Output: hELLO, wORLD!*

1. *string.count(substring): Returns the number of occurrences of the specified substring in the string.*

*Python*

*my\_string = "hello, world"*

*count = my\_string.count("o")*

*print(count) # Output: 3*

1. *string.find(substring): Returns the index of the first occurrence of the specified substring in the string. If not found, returns -1.*

*Python*

*my\_string = "hello, world"*

*index = my\_string.find("world")*

*print(index) # Output: 7*

1. *string.replace(old, new): Replaces all occurrences of the old substring with the new substring.*

*Python*

*my\_string = "hello, world"*

*new\_string = my\_string.replace("world", "everyone")*

*print(new\_string) # Output: hello, everyone*

*These attributes provide a variety of operations for manipulating and analyzing string data in Python.*

***Task 08:***

***A Python program encounters 3 general types of errors i.e., syntax errors,***

***logic errors and semantic errors. Briefly define each of these errors, their***

***reasons and how to avoid these errors.***

***Solution:***

*Types of Errors in Python*

*1. Syntax Errors*

*Definition: These errors occur when the code violates Python's grammar rules. They are typically caused by typos, missing punctuation, or incorrect indentation.Reasons:*

* *Typos: Incorrect spelling of keywords, variables, or functions.*
* *Missing punctuation: Omitting necessary symbols like parentheses, brackets, or quotation marks.*
* *Incorrect indentation: Indentation is crucial for Python's code blocks. Incorrect indentation can lead to syntax errors.How to Avoid:*
* *Follow Python's syntax rules carefully.*
* *Use a code editor or IDE that highlights syntax errors.*
* *Test your code frequently and thoroughly.*

*2. Logic Errors*

*Definition: These errors occur when the code runs without raising any exceptions but produces incorrect results due to flaws in the algorithm or logic.Reasons:*

* *Incorrect calculations or comparisons.*
* *Infinite loops or unintended recursion.*
* *Incorrect variable usage or assignments.*
* *Missing or extra conditions in conditional statements.How to Avoid:*
* *Plan and design your code carefully before writing it.*
* *Use debugging tools to step through your code and inspect variable values.*
* *Test your code with various inputs to identify unexpected behavior.*
* *Write clear and concise code that is easy to understand.*

*3. Semantic Errors*

*Definition: These errors occur when the code executes without errors but produces unexpected or unintended results due to a misunderstanding of the problem or incorrect implementation.Reasons:*

* *Incorrect interpretation of requirements.*
* *Missing or incorrect assumptions.*
* *Poor problem-solving skills.*
* *Lack of domain knowledge.How to Avoid:*
* *Clearly understand the problem and requirements.*
* *Break down the problem into smaller, manageable subproblems.*
* *Use appropriate algorithms and data structures.*
* *Test your code thoroughly with different scenarios.*
* *Seek clarification if you are unsure about any aspect of the problem.*

*By understanding these types of errors and following the suggested prevention techniques, you can significantly improve the quality and reliability of your Python code.*

***Task 09:***

***In Python programming, a conditional statement allows blocks of code to be***

***executed at runtime based on a condition. Define and describe different***

***conditional statements with examples.***

***Solution:***

*Conditional Statements in Python:*

*Conditional statements are essential programming constructs that enable your code to make decisions and execute different code blocks based on specific conditions. They allow you to create more dynamic and flexible programs.*

*Types of Conditional Statements*

1. *if Statement:*
   * *Executes a block of code if a condition is true.*
   * *Syntax:*

*Python*

*if condition:*

*# Code to be executed if condition is true*

* + *Example:*

*Python*

*age = 25*

*if age >= 18:*

*print("You are an adult.")*

1. *if-else Statement:*
   * *Executes one block of code if a condition is true, and another block if it's false.*
   * *Syntax:*

*Python*

*if condition:*

*# Code to be executed if condition is true*

*else:*

*# Code to be executed if condition is false*

* + *Example:*

*Python*

*x = 10*

*y = 20*

*if x > y:*

*print("x is greater than y.")*

*else:*

*print("y is greater than or equal to x.")*

1. *if-elif-else Statement:*
   * *Provides multiple conditions to check.*
   * *Executes the first block whose condition is true.*
   * *If no conditions are true, executes the else block.*
   * *Syntax:*

*if condition1: # Code to be executed if condition1 is true elif condition2: # Code to be executed if condition2 is true elif condition3: # Code to be executed if condition3 is true else: # Code to be executed if none of the conditions are true*

*- Example:*

*```python*

*grade = 85*

*if grade >= 90:*

*print("Excellent")*

*elif grade >= 80:*

*print("Very Good")*

*elif grade >= 70:*

*print("Good")*

*else:*

*print("Needs Improvement")*

*Nested Conditional Statements:*

* *You can nest conditional statements within other conditional statements for more complex decision-making.*
* *Example:*

*Python*

*age = 25*

*is\_student = True*

*if age >= 18:*

*if is\_student:*

*print("You are an adult student.")*

*else:*

*print("You are an adult non-student.")*

*else:*

*print("You are a   minor.")*

*By effectively using conditional statements, you can create programs that adapt to different situations and make informed decisions based on input data.*

***Task 10:***

***Boolean expression is a very useful mechanism in Python programming.***

***Define Boolean expression, describe its purpose and give examples of***

***comparison operators in Python.***

***Solution:***

*Boolean Expressions in Python*

*Definition: A Boolean expression is an expression that evaluates to either True or False. It is a fundamental building block of conditional statements and control flow in Python.*

*Purpose: Boolean expressions are used to make decisions and control the execution of code. They allow you to:*

* *Check conditions and perform actions based on the outcome.*
* *Create logical expressions to combine multiple conditions.*
* *Implement control flow structures like if, else, elif, while, and for loops.*

*Comparison Operators: Comparison operators are used to compare values and return a Boolean result. Here are the common comparison operators in Python:*

|  |  |  |
| --- | --- | --- |
| *Operator* | *Meaning* | *Example* |
| *==* | *Equal to* | *x == y* |
| *!=* | *Not equal to* | *x != y* |
| *<* | *Less than* | *x < y* |
| *<=* | *Less than or equal to* | *x <= y* |
| *>* | *Greater than* | *x > y* |
| *>=* | *Greater than or equal to* | *x >= y* |

*Examples:*

*Python*

*# Comparison operators*

*x = 10*

*y = 5*

*# Equality*

*is\_equal = x == y*

*print(is\_equal) # Output: False*

*# Inequality*

*is\_not\_equal = x != y*

*print(is\_not\_equal) # Output: True*

*# Less than*

*is\_less = x < y*

*print(is\_less) # Output: False*

*# Less than or equal to*

*is\_less\_or\_equal = x <= y*

*print(is\_less\_or\_equal) # Output: False*

*# Greater than*

*is\_greater = x > y*

*print(is\_greater) # Output: True*

*# Greater than or equal to*

*is\_greater\_or\_equal = x >= y*

*print(is\_greater\_or\_equal) # Output: True*

*Boolean expressions can be combined using logical operators (and, or, not) to create more complex conditions. For example:*

*Python*

*age = 25*

*is\_student = True*

*is\_adult\_student = age >= 18 and is\_student*

*print(is\_adult\_student) # Output: True*

*Boolean expressions are essential for writing effective and flexible Python code. They provide the foundation for decision-making and control flow in your programs.*