PYTHON NOTES

**UNIT I**

**INTRODUCTION TO PYTHON**

**BASIC SYNTAX**

**DATA TYPES**

**VARIABLES**

**OPERATORS**

**INPUT/OUTPUT**

**FLOW OF CONTROL**

**MODULES**

It looks like you’re interested in learning Python! Here’s a brief overview of each topic:

**1. Introduction to Python**

Python is a high-level, interpreted programming language known for its readability and simplicity. It’s widely used in web development, data analysis, artificial intelligence, scientific computing, and more.

**2. Basic Syntax**

Python’s syntax is clean and easy to understand. For example, to print “Hello, World!” you simply write:

print("Hello, World!")

**3. Data Types**

Python supports various data types, including:

* **Integers**: Whole numbers, e.g., 5
* **Floats**: Decimal numbers, e.g., 3.14
* **Strings**: Text, e.g., "Hello"
* **Lists**: Ordered collections, e.g., [1, 2, 3]
* **Dictionaries**: Key-value pairs, e.g., {"name": "Alice", "age": 25}

**4. Variables**

Variables are used to store data. You can create a variable by assigning a value to a name:

x = 10

name = "Alice"

**5. Operators**

Operators are used to perform operations on variables and values. Common operators include:

* **Arithmetic Operators**: +, -, \*, /
* **Comparison Operators**: ==, !=, >, <
* **Logical Operators**: and, or, not

**6. Input/Output**

You can take input from the user and display output using the input() and print() functions:

name = input("Enter your name: ")

print("Hello, " + name)

**7. Flow of Control**

Control flow statements like if, for, and while are used to control the execution of code:

if x > 0:

print("Positive")

else:

print("Non-positive")

for i in range(5):

print(i)

while x > 0:

print(x)

x -= 1

**8. Modules**

Modules are files containing Python code that can be imported and used in other Python programs. For example, to use the math module:

import math

print(math.sqrt(16))

Would you like to dive deeper into any of these topics?

**BRANCHING**

**IF**

**IF-ELSE**

**NESTED IF-ELSE**

**LOOPING**

**FOR**

**WHILE**

**NESTED LOOPS**

**CONTROL STRUCTURES**

**BREAK**

**CONTINUE**

**PASS**

Sure! Let’s dive into these concepts in Python:

### Branching

#### If Statement

The if statement allows you to execute a block of code only if a certain condition is true.

x = 10

if x > 5:

print("x is greater than 5")

#### If-Else Statement

The if-else statement lets you execute one block of code if a condition is true, and another block if it is false.

x = 10

if x > 5:

print("x is greater than 5")

else:

print("x is not greater than 5")

#### Nested If-Else

You can nest if-else statements to check multiple conditions.

x = 10

if x > 5:

if x > 8:

print("x is greater than 8")

else:

print("x is between 6 and 8")

else:

print("x is 5 or less")

### Looping

#### For Loop

The for loop is used to iterate over a sequence (like a list, tuple, or string).

for i in range(5):

print(i)

#### While Loop

The while loop continues to execute as long as a condition is true.

x = 5

while x > 0:

print(x)

x -= 1

#### Nested Loops

You can nest loops inside each other.

for i in range(3):

for j in range(2):

print(f"i: {i}, j: {j}")

### Control Structures

#### Break

The break statement exits the loop prematurely.

for i in range(5):

if i == 3:

break

print(i)

#### Continue

The continue statement skips the current iteration and continues with the next one.

for i in range(5):

if i == 3:

continue

print(i)

#### Pass

The pass statement does nothing and is used as a placeholder.

for i in range(5):

if i == 3:

pass

print(i)

Would you like to see more examples or explanations on any specific topic?

**STRINGS AND TUPLES**

**ACCESSING STRINGS**

**BASIC OPERATIONS**

**STRING SLICES**

**WORKING WITH LISTS**

**INTRODUCTION**

**ACCESSING LIST OPERATIONS**

Sure! Let’s go through each of these topics in Python:

### Strings and Tuples

#### Accessing Strings

Strings in Python are sequences of characters. You can access individual characters using indexing:

s = "Hello, World!"

print(s[0]) # Output: H

print(s[-1]) # Output: !

#### Basic Operations

You can perform various operations on strings, such as concatenation, repetition, and finding the length:

s1 = "Hello"

s2 = "World"

print(s1 + " " + s2) # Output: Hello World

print(s1 \* 3) # Output: HelloHelloHello

print(len(s1)) # Output: 5

#### String Slices

String slicing allows you to get a substring by specifying a start and end index:

s = "Hello, World!"

print(s[0:5]) # Output: Hello

print(s[7:]) # Output: World!

print(s[:5]) # Output: Hello

print(s[::2]) # Output: Hlo ol!

### Working with Lists

#### Introduction

Lists are ordered collections of items that can be of different types. They are mutable, meaning you can change their content.

#### Accessing List Elements

You can access list elements using indexing and slicing, similar to strings:

lst = [1, 2, 3, 4, 5]

print(lst[0]) # Output: 1

print(lst[-1]) # Output: 5

print(lst[1:3]) # Output: [2, 3]

#### List Operations

Lists support various operations like appending, removing, and sorting:

lst = [1, 2, 3]

lst.append(4) # Adds 4 to the end of the list

print(lst) # Output: [1, 2, 3, 4]

lst.remove(2) # Removes the first occurrence of 2

print(lst) # Output: [1, 3, 4]

lst.sort(reverse=True) # Sorts the list in descending order

print(lst) # Output: [4, 3, 1]

### Tuples

Tuples are similar to lists but are immutable, meaning their content cannot be changed after creation.

#### Creating and Accessing Tuples

You can create tuples using parentheses and access their elements using indexing:

tup = (1, 2, 3, "Hello")

print(tup[0]) # Output: 1

print(tup[-1]) # Output: Hello

#### Basic Operations

Tuples support operations like concatenation and repetition:

tup1 = (1, 2, 3)

tup2 = (4, 5, 6)

print(tup1 + tup2) # Output: (1, 2, 3, 4, 5, 6)

print(tup1 \* 2) # Output: (1, 2, 3, 1, 2, 3)

Would you like to explore any of these topics in more detail?

**UNIT II**

**FUNCTON AND METHODS**

**FILES**

**MODULES**

**DICTIONARIES**

**FUNCTIONS**

**FUNCTIONAL PROGRAMMING**

**DECLARING AND CALLING FUNCTIONS**

**DECLARE**

Sure! Let’s go through each of these topics in Python:

### Functions and Methods

#### Declaring and Calling Functions

Functions are blocks of code that perform specific tasks. You can define a function using the def keyword and call it by its name.

def greet(name):

return f"Hello, {name}!"

print(greet("Alice")) # Output: Hello, Alice!

### Files

#### Reading and Writing Files

You can read from and write to files using Python’s built-in functions.

# Writing to a file

with open("example.txt", "w") as file:

file.write("Hello, World!")

# Reading from a file

with open("example.txt", "r") as file:

content = file.read()

print(content) # Output: Hello, World!

### Modules

#### Importing Modules

Modules are files containing Python code that can be imported into other Python programs.

import math

print(math.sqrt(16)) # Output: 4.0

### Dictionaries

#### Creating and Accessing Dictionaries

Dictionaries are collections of key-value pairs.

# Creating a dictionary

person = {"name": "Alice", "age": 25}

# Accessing values

print(person["name"]) # Output: Alice

print(person.get("age")) # Output: 25

### Functional Programming

#### Using Functions as First-Class Citizens

In Python, functions can be passed as arguments to other functions, returned as values, and assigned to variables.

def add(x, y):

return x + y

def operate(func, x, y):

return func(x, y)

print(operate(add, 5, 3)) # Output: 8

### Declaring and Calling Functions

#### Declaring Functions

You declare a function using the def keyword, followed by the function name and parameters.

def multiply(a, b):

return a \* b

# Calling the function

result = multiply(4, 5)

print(result) # Output: 20

Would you like to explore any of these topics in more detail or see more examples?

**UNIT III**

**FILE HANDLING**

**INTRODUCTION**

**FILE TYPES**

**CREATING FILES**

**OPENING FILES**

**CLOSING FILES**

**RENAMING FILES**

**ACCESSING FILES**

**DELETING FILES**

**FILE POINTERS**

**FILE MODES**

**BINARY FILES**

**EXCEPTION HANDLING**

**INTRODUCTION**

**EXCEPTION**

**DEFAULT EXCEPTION HANDLERS**

**HANDLING MULTIPLE ERRORS**

**RAISING EXCEPTIONS**

**GENERATORS**

**PYTHON LIBRARIES**

**UNIT IV**

**DATABASE INTERACTIONS**

**SQL DATABASE CONNECTION USING PYTHON**

**CREATING AND SEARCHING TABLES**

**READING AND STORING CONFIG INFORMATION ON A DATABASE**

**PROGRAMMING USING DATABASE CONNECTIONS**

**PYTHON MULTITHREADING**

**UNDERSTANDING THREADS**

**FORKING THREADS**

**SYNCHRONIZING THE THREADS**

**PROGRAMMING USING MULTITHREADING**

**UNIT V**

**INTRODUCTION TO MACJINE LEARNING PACKAGES**

**NUMPY**

**SCIPY**

**PANDAS**

**EXPLAIN MUTABLE AND IMMUTABLE DATA TYPES IN PYTHON**

Sure! In Python, data types can be classified as either **mutable** or **immutable**. This classification is based on whether the value of an object can be changed after it is created.

**Mutable Data Types**

Mutable objects can be modified after their creation. This means you can change their content without changing their identity. Here are some common mutable data types in Python:

* **Lists**: You can add, remove, or change elements.
* my\_list = [1, 2, 3]
* my\_list.append(4) # my\_list is now [1, 2, 3, 4]
* my\_list[0] = 10 # my\_list is now [10, 2, 3, 4]
* **Dictionaries**: You can add, remove, or change key-value pairs.
* my\_dict = {'a': 1, 'b': 2}
* my\_dict['c'] = 3 # my\_dict is now {'a': 1, 'b': 2, 'c': 3}
* my\_dict['a'] = 10 # my\_dict is now {'a': 10, 'b': 2, 'c': 3}
* **Sets**: You can add or remove elements.
* my\_set = {1, 2, 3}
* my\_set.add(4) # my\_set is now {1, 2, 3, 4}
* my\_set.remove(2) # my\_set is now {1, 3, 4}

**Immutable Data Types**

Immutable objects cannot be modified after their creation. Any attempt to change their value results in the creation of a new object. Here are some common immutable data types in Python:

* **Strings**: Any modification creates a new string.
* my\_string = "hello"
* new\_string = my\_string.replace('h', 'j') # new\_string is "jello", my\_string remains "hello"
* **Tuples**: You cannot change their elements.
* my\_tuple = (1, 2, 3)
* # my\_tuple[0] = 10 # This will raise a TypeError
* **Numbers**: Integers, floats, and complex numbers are immutable.
* x = 10
* y = x + 5 # y is 15, x remains 10

Understanding the difference between mutable and immutable data types is crucial for writing efficient and bug-free code. [Mutable objects are useful when you need to change the content frequently, while immutable objects are safer to use when you want to ensure the data remains constant1](https://realpython.com/python-mutable-vs-immutable-types/)[2](https://www.geeksforgeeks.org/mutable-vs-immutable-objects-in-python/)[3](https://www.javatpoint.com/python-mutable-vs-immutable-data-types).

Do you have any specific questions about these data types?

**DISCUSS THE VARIOUS OPERATORS IN PYTHON**

Ce Certainly! Python has a variety of operators that allow you to perform different operations on variables and values. Here’s a breakdown of the main types of operators in Python:

**1. Arithmetic Operators**

These operators are used to perform common mathematical operations:

* **Addition (+)**: Adds two operands.
* x = 5 + 3 # x is 8
* **Subtraction (-)**: Subtracts the second operand from the first.
* x = 5 - 3 # x is 2
* **Multiplication (\*)**: Multiplies two operands.
* x = 5 \* 3 # x is 15
* **Division (/)**: Divides the first operand by the second.
* x = 5 / 3 # x is 1.666...
* **Modulus (%)**: Returns the remainder of the division.
* x = 5 % 3 # x is 2
* **Exponentiation (\*\*)**: Raises the first operand to the power of the second.
* x = 5 \*\* 3 # x is 125
* **Floor Division (//)**: Divides and returns the largest integer less than or equal to the result.
* x = 5 // 3 # x is 1

**2. Assignment Operators**

These operators are used to assign values to variables:

* **Assignment (=)**: Assigns the right-hand operand to the left-hand operand.
* x = 5
* **Add and Assign (+=)**: Adds the right-hand operand to the left-hand operand and assigns the result to the left-hand operand.
* x += 3 # x is now 8 if x was 5
* **Subtract and Assign (-=)**: Subtracts the right-hand operand from the left-hand operand and assigns the result to the left-hand operand.
* x -= 3 # x is now 2 if x was 5
* **Multiply and Assign (\*=)**: Multiplies the right-hand operand with the left-hand operand and assigns the result to the left-hand operand.
* x \*= 3 # x is now 15 if x was 5
* **Divide and Assign (/=)**: Divides the left-hand operand by the right-hand operand and assigns the result to the left-hand operand.
* x /= 3 # x is now 1.666... if x was 5

**3. Comparison Operators**

These operators compare two values and return a boolean result (True or False):

* **Equal to (==)**: Checks if two operands are equal.
* x == y
* **Not equal to (!=)**: Checks if two operands are not equal.
* x != y
* **Greater than (>)**: Checks if the left operand is greater than the right.
* x > y
* **Less than (<)**: Checks if the left operand is less than the right.
* x < y
* **Greater than or equal to (>=)**: Checks if the left operand is greater than or equal to the right.
* x >= y
* **Less than or equal to (<=)**: Checks if the left operand is less than or equal to the right.
* x <= y

**4. Logical Operators**

These operators are used to combine conditional statements:

* **AND (and)**: Returns True if both statements are true.
* x < 5 and x < 10
* **OR (or)**: Returns True if one of the statements is true.
* x < 5 or x < 4
* **NOT (not)**: Reverses the result, returns False if the result is true.
* not(x < 5 and x < 10)

**5. Identity Operators**

These operators compare the memory locations of two objects:

* **IS (is)**: Returns True if both variables are the same object.
* x is y
* **IS NOT (is not)**: Returns True if both variables are not the same object.
* x is not y

**6. Membership Operators**

These operators test for membership in a sequence (like strings, lists, or tuples):

* **IN (in)**: Returns True if a sequence with the specified value is present in the object.
* x in y
* **NOT IN (not in)**: Returns True if a sequence with the specified value is not present in the object.
* x not in y

**7. Bitwise Operators**

These operators are used to compare (binary) numbers:

* **AND (&)**: Sets each bit to 1 if both bits are 1.
* x & y
* **OR (|)**: Sets each bit to 1 if one of two bits is 1.
* x | y
* **XOR (^)**: Sets each bit to 1 if only one of two bits is 1.
* x ^ y
* **NOT (~)**: Inverts all the bits.
* ~x
* **Zero fill left shift (<<)**: Shift left by pushing zeros in from the right and let the leftmost bits fall off.
* x << 2
* **Signed right shift (>>)**: Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off.
* x >> 2

[These operators form the foundation of many operations in Python programming1](https://www.w3schools.com/python/python_operators.asp)[2](https://www.programiz.com/python-programming/operators)[3](https://realpython.com/python-operators-expressions/). If you have any specific questions or need examples, feel free to ask!

rtainly! Python has a variety of operators that allow you to perform different operations on variables and values. Here’s a breakdown of the main types of operators in Python:

**1. Arithmetic Operators**

These operators are used to perform common mathematical operations:

* **Addition (+)**: Adds two operands.
* x = 5 + 3 # x is 8
* **Subtraction (-)**: Subtracts the second operand from the first.
* x = 5 - 3 # x is 2
* **Multiplication (\*)**: Multiplies two operands.
* x = 5 \* 3 # x is 15
* **Division (/)**: Divides the first operand by the second.
* x = 5 / 3 # x is 1.666...
* **Modulus (%)**: Returns the remainder of the division.
* x = 5 % 3 # x is 2
* **Exponentiation (\*\*)**: Raises the first operand to the power of the second.
* x = 5 \*\* 3 # x is 125
* **Floor Division (//)**: Divides and returns the largest integer less than or equal to the result.
* x = 5 // 3 # x is 1

**2. Assignment Operators**

These operators are used to assign values to variables:

* **Assignment (=)**: Assigns the right-hand operand to the left-hand operand.
* x = 5
* **Add and Assign (+=)**: Adds the right-hand operand to the left-hand operand and assigns the result to the left-hand operand.
* x += 3 # x is now 8 if x was 5
* **Subtract and Assign (-=)**: Subtracts the right-hand operand from the left-hand operand and assigns the result to the left-hand operand.
* x -= 3 # x is now 2 if x was 5
* **Multiply and Assign (\*=)**: Multiplies the right-hand operand with the left-hand operand and assigns the result to the left-hand operand.
* x \*= 3 # x is now 15 if x was 5
* **Divide and Assign (/=)**: Divides the left-hand operand by the right-hand operand and assigns the result to the left-hand operand.
* x /= 3 # x is now 1.666... if x was 5

**3. Comparison Operators**

These operators compare two values and return a boolean result (True or False):

* **Equal to (==)**: Checks if two operands are equal.
* x == y
* **Not equal to (!=)**: Checks if two operands are not equal.
* x != y
* **Greater than (>)**: Checks if the left operand is greater than the right.
* x > y
* **Less than (<)**: Checks if the left operand is less than the right.
* x < y
* **Greater than or equal to (>=)**: Checks if the left operand is greater than or equal to the right.
* x >= y
* **Less than or equal to (<=)**: Checks if the left operand is less than or equal to the right.
* x <= y

**4. Logical Operators**

These operators are used to combine conditional statements:

* **AND (and)**: Returns True if both statements are true.
* x < 5 and x < 10
* **OR (or)**: Returns True if one of the statements is true.
* x < 5 or x < 4
* **NOT (not)**: Reverses the result, returns False if the result is true.
* not(x < 5 and x < 10)

**5. Identity Operators**

These operators compare the memory locations of two objects:

* **IS (is)**: Returns True if both variables are the same object.
* x is y
* **IS NOT (is not)**: Returns True if both variables are not the same object.
* x is not y

**6. Membership Operators**

These operators test for membership in a sequence (like strings, lists, or tuples):

* **IN (in)**: Returns True if a sequence with the specified value is present in the object.
* x in y
* **NOT IN (not in)**: Returns True if a sequence with the specified value is not present in the object.
* x not in y

**7. Bitwise Operators**

These operators are used to compare (binary) numbers:

* **AND (&)**: Sets each bit to 1 if both bits are 1.
* x & y
* **OR (|)**: Sets each bit to 1 if one of two bits is 1.
* x | y
* **XOR (^)**: Sets each bit to 1 if only one of two bits is 1.
* x ^ y
* **NOT (~)**: Inverts all the bits.
* ~x
* **Zero fill left shift (<<)**: Shift left by pushing zeros in from the right and let the leftmost bits fall off.
* x << 2
* **Signed right shift (>>)**: Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off.
* x >> 2

[These operators form the foundation of many operations in Python programming1](https://www.w3schools.com/python/python_operators.asp)[2](https://www.programiz.com/python-programming/operators)[3](https://realpython.com/python-operators-expressions/). If you have any specific questions or need examples, feel free to ask!

**ILLUSTRATE THE DIFFERENT TYPES OF CONTROL FLOW STATEMENTS AVAILABLE IN PYTHON**

Sure! Control flow statements in Python allow you to dictate the order in which code is executed. They enable decision-making, looping, and altering the flow of execution. Here are the main types of control flow statements in Python:

**1. Conditional Statements**

These statements execute code blocks based on certain conditions.

* **if statement**: Executes a block of code if a specified condition is true.
* if condition:
* # code block

Example:

x = 10

if x > 5:

print("x is greater than 5")

* **if-else statement**: Executes one block of code if the condition is true, and another block if the condition is false.
* if condition:
* # code block if condition is true
* else:
* # code block if condition is false

Example:

x = 10

if x > 15:

print("x is greater than 15")

else:

print("x is not greater than 15")

* **if-elif-else statement**: Checks multiple conditions in sequence.
* if condition1:
* # code block if condition1 is true
* elif condition2:
* # code block if condition2 is true
* else:
* # code block if none of the conditions are true

Example:

x = 10

if x > 15:

print("x is greater than 15")

elif x > 5:

print("x is greater than 5 but not greater than 15")

else:

print("x is 5 or less")

**2. Looping Statements**

These statements execute a block of code repeatedly as long as a condition is true.

* **for loop**: Iterates over a sequence (like a list, tuple, or string).
* for item in sequence:
* # code block

Example:

for i in range(5):

print(i)

* **while loop**: Repeats a block of code as long as a condition is true.
* while condition:
* # code block

Example:

x = 0

while x < 5:

print(x)

x += 1

**3. Transfer Statements**

These statements alter the flow of execution in a loop or conditional block.

* **break**: Exits the nearest enclosing loop.
* for i in range(10):
* if i == 5:
* break
* print(i)
* **continue**: Skips the rest of the code inside the loop for the current iteration and moves to the next iteration.
* for i in range(10):
* if i == 5:
* continue
* print(i)
* **pass**: Does nothing and is used as a placeholder.
* for i in range(10):
* if i == 5:
* pass
* print(i)

**4. Exception Handling Statements**

These statements handle exceptions (errors) that occur during execution.

* **try-except**: Catches and handles exceptions.
* try:
* # code block that may raise an exception
* except ExceptionType:
* # code block to handle the exception

Example:

try:

x = 1 / 0

except ZeroDivisionError:

print("Cannot divide by zero")

* **finally**: Executes a block of code no matter what, after try and except blocks.
* try:
* # code block that may raise an exception
* except ExceptionType:
* # code block to handle the exception
* finally:
* # code block that will always execute

Example:

try:

x = 1 / 0

except ZeroDivisionError:

print("Cannot divide by zero")

finally:

print("This will always execute")

[These control flow statements are essential for creating dynamic and responsive programs1](https://pynative.com/python-control-flow-statements/)[2](https://www.leetpython.com/docs/The-1hr-Guide-To-Python/Programming%20Fundamentals/Control%20Flow%20Statements)[3](https://www.geeksforgeeks.org/python-if-else/). If you have any specific questions or need further examples, feel free to ask!

**MENTION VARIOUS METHODS OF LIST IN PYTHON**

Explore

Sure! Python lists come with a variety of built-in methods that allow you to perform different operations. Here are some of the most commonly used list methods:

1. **append()**: Adds an element to the end of the list.
2. my\_list = [1, 2, 3]
3. my\_list.append(4) # my\_list is now [1, 2, 3, 4]
4. **extend()**: Adds all elements of an iterable (like another list) to the end of the list.
5. my\_list = [1, 2, 3]
6. my\_list.extend([4, 5]) # my\_list is now [1, 2, 3, 4, 5]
7. **insert()**: Inserts an element at a specified position.
8. my\_list = [1, 2, 3]
9. my\_list.insert(1, 'a') # my\_list is now [1, 'a', 2, 3]
10. **remove()**: Removes the first occurrence of a specified element.
11. my\_list = [1, 2, 3, 2]
12. my\_list.remove(2) # my\_list is now [1, 3, 2]
13. **pop()**: Removes and returns the element at a specified position (default is the last element).
14. my\_list = [1, 2, 3]
15. my\_list.pop() # returns 3, my\_list is now [1, 2]
16. **clear()**: Removes all elements from the list.
17. my\_list = [1, 2, 3]
18. my\_list.clear() # my\_list is now []
19. **index()**: Returns the index of the first occurrence of a specified element.
20. my\_list = [1, 2, 3]
21. my\_list.index(2) # returns 1
22. **count()**: Returns the number of occurrences of a specified element.
23. my\_list = [1, 2, 2, 3]
24. my\_list.count(2) # returns 2
25. **sort()**: Sorts the list in ascending order by default (can be customized with parameters).
26. my\_list = [3, 1, 2]
27. my\_list.sort() # my\_list is now [1, 2, 3]
28. **reverse()**: Reverses the order of the list.
29. my\_list = [1, 2, 3]
30. my\_list.reverse() # my\_list is now [3, 2, 1]
31. **copy()**: Returns a shallow copy of the list.
32. my\_list = [1, 2, 3]
33. new\_list = my\_list.copy() # new\_list is [1, 2, 3]

[These methods provide a lot of flexibility and functionality for working with lists in Python1](https://bing.com/search?q=methods+of+list+in+Python)[2](https://www.w3schools.com/python/python_lists_methods.asp)[3](https://www.geeksforgeeks.org/list-methods-python/). If you need more details or examples on any specific method, feel free to ask!

**MENTION VARIOUS METHODS OF STRINGS IN PYTHON**

Certainly! Python provides a rich set of methods for manipulating and working with strings. Here are some of the most commonly used string methods:

1. **capitalize()**: Converts the first character to uppercase.
2. text = "hello"
3. print(text.capitalize()) # Output: "Hello"
4. **casefold()**: Converts the string to lowercase, more aggressive than lower().
5. text = "HELLO"
6. print(text.casefold()) # Output: "hello"
7. **center(width)**: Centers the string in a field of given width.
8. text = "hello"
9. print(text.center(10)) # Output: " hello "
10. **count(substring)**: Returns the number of occurrences of a substring.
11. text = "hello world"
12. print(text.count("o")) # Output: 2
13. **encode()**: Encodes the string using the specified encoding.
14. text = "hello"
15. print(text.encode()) # Output: b'hello'
16. **endswith(suffix)**: Checks if the string ends with the specified suffix.
17. text = "hello"
18. print(text.endswith("o")) # Output: True
19. **find(substring)**: Returns the lowest index of the substring.
20. text = "hello"
21. print(text.find("e")) # Output: 1
22. **format()**: Formats specified values in a string.
23. text = "Hello, {}"
24. print(text.format("world")) # Output: "Hello, world"
25. **index(substring)**: Returns the lowest index of the substring, raises an error if not found.
26. text = "hello"
27. print(text.index("e")) # Output: 1
28. **isalnum()**: Checks if all characters are alphanumeric.
29. text = "hello123"
30. print(text.isalnum()) # Output: True
31. **isalpha()**: Checks if all characters are alphabetic.
32. text = "hello"
33. print(text.isalpha()) # Output: True
34. **isdigit()**: Checks if all characters are digits.
35. text = "123"
36. print(text.isdigit()) # Output: True
37. **islower()**: Checks if all characters are lowercase.
38. text = "hello"
39. print(text.islower()) # Output: True
40. **isspace()**: Checks if all characters are whitespace.
41. text = " "
42. print(text.isspace()) # Output: True
43. **istitle()**: Checks if the string follows the rules of a title.
44. text = "Hello World"
45. print(text.istitle()) # Output: True
46. **isupper()**: Checks if all characters are uppercase.
47. text = "HELLO"
48. print(text.isupper()) # Output: True
49. **join(iterable)**: Joins elements of an iterable with the string as a separator.
50. text = "-"
51. print(text.join(["hello", "world"])) # Output: "hello-world"
52. **lower()**: Converts all characters to lowercase.
53. text = "HELLO"
54. print(text.lower()) # Output: "hello"
55. **replace(old, new)**: Replaces occurrences of a substring with another substring.
56. text = "hello world"
57. print(text.replace("world", "Python")) # Output: "hello Python"
58. **split(separator)**: Splits the string at the specified separator and returns a list.
59. text = "hello world"
60. print(text.split()) # Output: ["hello", "world"]
61. **strip()**: Removes leading and trailing whitespace.
62. text = " hello "
63. print(text.strip()) # Output: "hello"
64. **upper()**: Converts all characters to uppercase.
65. text = "hello"
66. print(text.upper()) # Output: "HELLO"

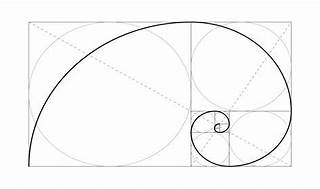
[These methods provide a wide range of functionalities for string manipulation in Python1](https://www.w3schools.com/python/python_ref_string.asp)[2](https://docs.python.org/3/library/string.html)[3](https://www.geeksforgeeks.org/python-string-methods/). If you need more details or examples on any specific method, feel free to ask!

**WRITE A PROGRAM TO DISPLAY THE FIBONACCI SEQUENCES IN PYTHON**

**WRITE A PROGRAM FOR A NESTED LOOP IN PYTHON**

**WRITE A PROGRAM TO CHECK WHETHER A NUMBER IS PRIME OR NOT IN PYTHON**

**WRITE A PYTHON PROGRAM FOR MARKS GRADE SYSTEM OUT OF 100 MARKS**



Explore

Sure! Here are the Python programs for each of your requests:

**1. Fibonacci Sequence**

This program displays the Fibonacci sequence up to a specified number of terms.

def fibonacci(n):

a, b = 0, 1

for \_ in range(n):

print(a, end=" ")

a, b = b, a + b

# Example usage

num\_terms = 10

fibonacci(num\_terms)

**2. Nested Loop**

This program demonstrates a nested loop by printing a multiplication table.

for i in range(1, 6): # Outer loop

for j in range(1, 6): # Inner loop

print(f"{i \* j:2}", end=" ")

print()

**3. Check Prime Number**

This program checks whether a given number is prime.

def is\_prime(n):

if n <= 1:

return False

for i in range(2, int(n\*\*0.5) + 1):

if n % i == 0:

return False

return True

# Example usage

number = 29

if is\_prime(number):

print(f"{number} is a prime number")

else:

print(f"{number} is not a prime number")

**4. Marks Grade System**

This program assigns grades based on marks out of 100.

def calculate\_grade(marks):

if marks >= 90:

return 'A'

elif marks >= 80:

return 'B'

elif marks >= 70:

return 'C'

elif marks >= 60:

return 'D'

else:

return 'F'

# Example usage

marks = float(input("Enter marks out of 100: "))

grade = calculate\_grade(marks)

print(f"Marks: {marks}, Grade: {grade}")

Feel free to run these programs and modify them as needed! If you have any questions or need further assistance, just let me know.