

Day1:

TASK :- Data Types in Python

Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.

```
>>> num = 2.5
>>> type(num)
<class 'float'>
>>> num = 5
>>> type(num)
<class 'int'>
>>> num = 6+9j
>>> type(num)
<class 'complex'>
>>> a = 5.6
>>> b = int(a)
>>> type(b)
<class 'int'>
>>> b
5
>>> k = float(b)
>>> k
5.0
>>>
```

Numeric

- int
- float
- complex
- bool

Navin Reddy

Subscribe Telusko

```
1 x = 9921004873-Gudala Guna Sankar Reddy
2 print(type(x))
3 x1 = str("Hello World")
4 x2 = complex(1j)
5 w = float("4.2")
6 a = "Hello"
7 print(a)
8 a1 = "Hello, World!"
9 print(len(a))
10 text = "The best things in life are free!"
11 print("free" in text)
12 b = "Hello, World!"
13 print(b[-5:-2])
14 a2 = "Hello, World"
15 print(a.strip()) # returns "Hello, World!"
16 a3 = "Hello!"
17 b4 = "World"
18 c5 = a + " " + b
19 print(c5)
20 age = 36
21 text = f"my name is John, I am {age}"
22 print(text)
23 price = 59
24 txt1 = f"The price is {price:.2f} dollars"
25 print(txt1)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
<class 'int'>
Hello
5
True
or!
Hello
Hello Hello, World!
My name is John, I am 36
My name is John, I am 36
The price is 59.00 dollars
```

Today, I delved into several fundamental concepts of Python, gaining a deeper understanding of its versatility. I explored **variable types**, the foundation of storing and manipulating data. Then, I studied **strings**, their **slicing**, and the **built-in methods** that enhance string manipulation, such as `.lower()`, `.upper()`, and `.replace()`. Additionally, I learned about **escape characters**, which enable seamless handling of special characters like newlines (`\n`) and quotes. Moving forward, I understood **Boolean values**, which represent truth values, and their crucial role in decision-making and logic building.

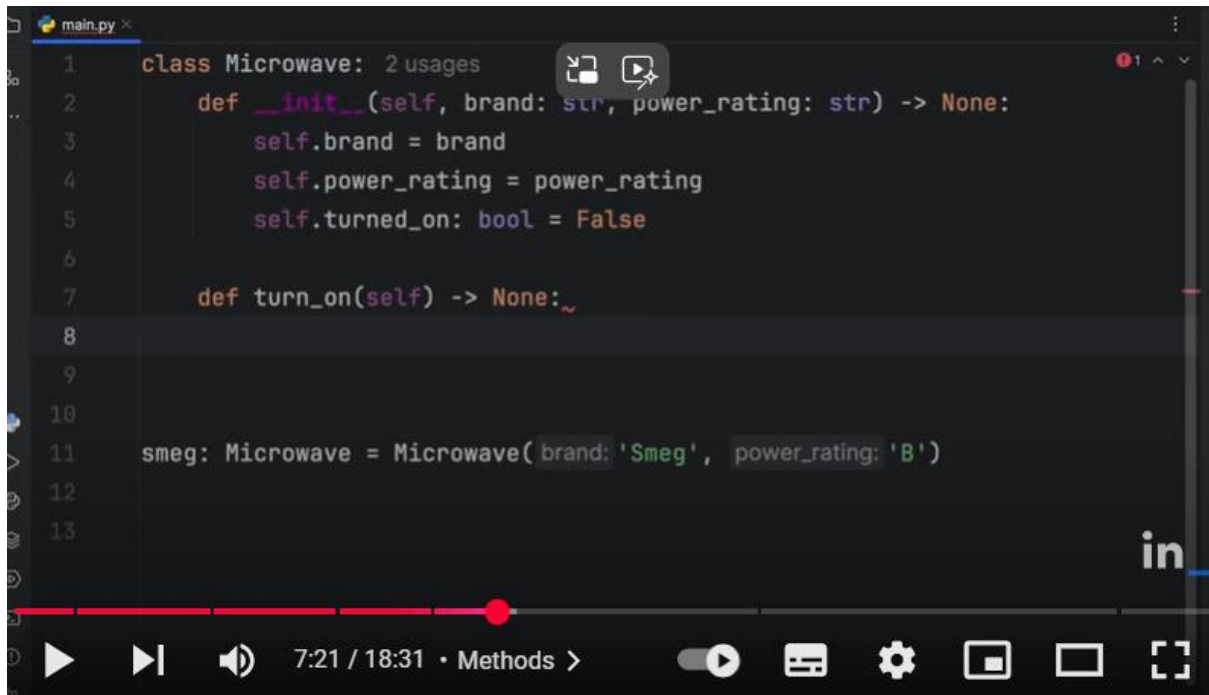
I also studied various **Python operators**, including arithmetic, comparison, logical, and membership operators, which are essential for performing operations and evaluations. Understanding **lists**, their mutability, and versatile operations, such as appending and slicing, added a practical dimension. Learning about **tuples**—immutable sequences—and **sets**, which allow unique, unordered collections,

broadened my knowledge of handling different data types. I also explored **dictionaries**, powerful data structures for key-value pairing, which simplify complex data organization.

Finally, I gained insights into **methods**, understanding how they add functionality to Python objects, and their ability to enhance code efficiency. Overall, these foundational topics have significantly enhanced my Python knowledge, preparing me to tackle more advanced programming challenges confidently.

Day2:

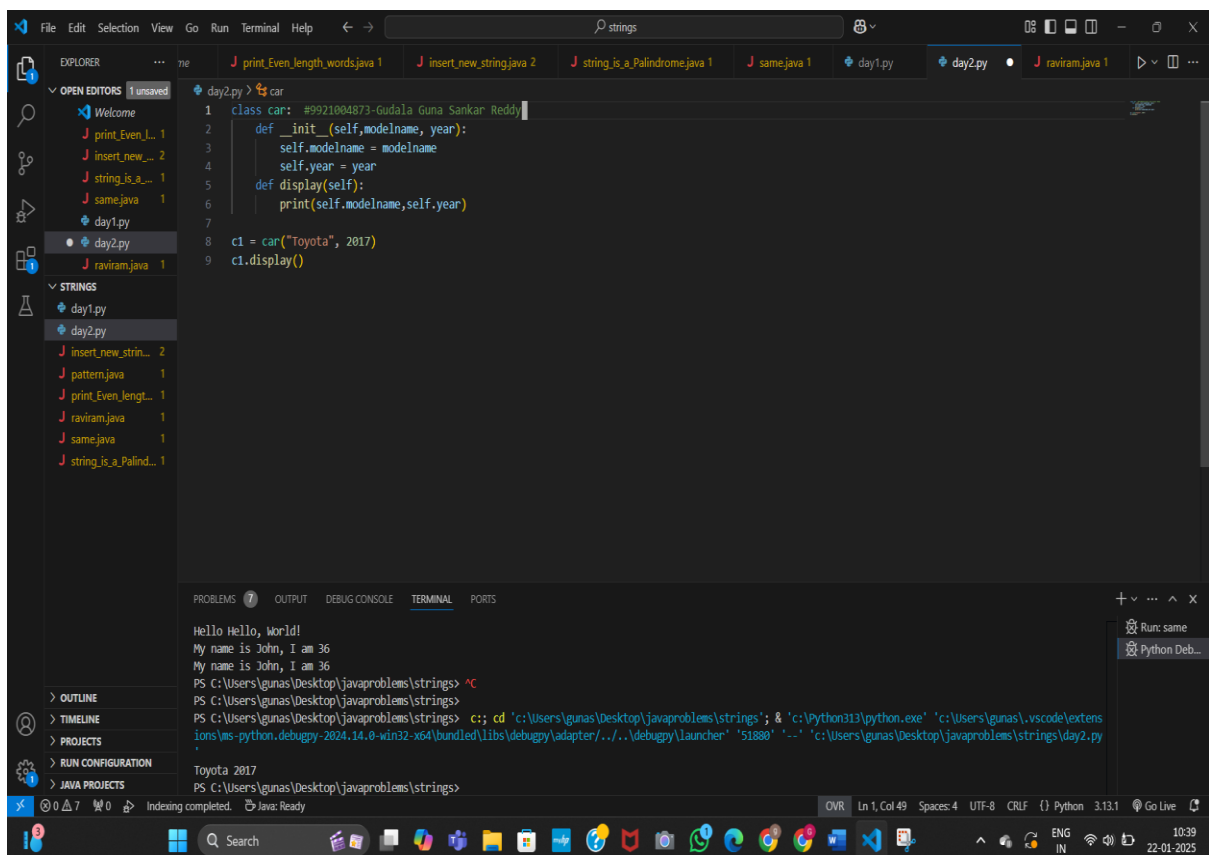
TASK :- OOPS



The screenshot shows a video player interface with a dark theme. The video content displays a Python code editor with the following code:

```
1 class Microwave: 2 usages
2     def __init__(self, brand: str, power_rating: str) -> None:
3         self.brand = brand
4         self.power_rating = power_rating
5         self.turned_on: bool = False
6
7     def turn_on(self) -> None:~
8
9
10
11 smeg: Microwave = Microwave(brand: 'Smeg', power_rating: 'B')
12
13
```

The video player controls at the bottom show a progress bar at 7:21 / 18:31, a volume icon, and a 'Methods' button.



The screenshot shows the Visual Studio Code (VS Code) IDE interface. The Explorer panel on the left shows a project structure with files like 'Welcome', 'print_Even_L...', 'insert_new...', 'string_is_a...', 'same.java', 'day1.py', 'day2.py', and 'raviram.java'. The main editor window shows a Python file named 'day2.py' with the following code:

```
1 class car: #9921004873-Gudala Guna Sankar Reddy
2     def __init__(self,modelname, year):
3         self.modelname = modelname
4         self.year = year
5     def display(self):
6         print(self.modelname,self.year)
7
8 c1 = car("Toyota", 2017)
9 c1.display()
```

The Output panel at the bottom shows the execution results:

```
Hello Hello, World!
My name is John, I am 36
My name is John, I am 36
PS C:\Users\gunas\Desktop\javaproblems\strings> ^C
PS C:\Users\gunas\Desktop\javaproblems\strings>
PS C:\Users\gunas\Desktop\javaproblems\strings> c:: cd 'c:\Users\gunas\Desktop\javaproblems\strings' & 'c:\Python313\python.exe' 'c:\Users\gunas\.vscode\extensions\ms-python.debugpy-2024.14.0-win32-x64\bundle\libs\debugpy\adapter\..\..\debugpy\launcher' '51880' '-...' 'c:\Users\gunas\Desktop\javaproblems\strings\day2.py'
Toyota 2017
PS C:\Users\gunas\Desktop\javaproblems\strings>
```

The status bar at the bottom indicates the current file is 'day2.py' at line 1, column 49, with 4 spaces, UTF-8 encoding, CRLF line endings, and Python 3.13.1 interpreter.

Today, I explored key concepts in object-oriented programming (OOP) using Python, which deepened my understanding of structuring and organizing code efficiently. I began by learning about **classes**, which serve as blueprints for creating objects. Following this, I studied **objects**, the specific instances of a class, and how they store data and behaviors defined by the class. Next, I delved into **methods**, the functions associated with a class, which define the actions objects can perform.

Moving on to more advanced topics, I learned about **inheritance**, which allows a class to derive properties and behaviors from another class, promoting reusability and reducing redundancy. I also explored **polymorphism**, the ability of objects to take on multiple forms, enabling a unified interface to handle different types dynamically. The concept of **data abstraction** was fascinating, as it focuses on exposing only the essential features of objects while hiding unnecessary details, simplifying complexity.

Lastly, I studied **encapsulation**, which enforces data security by restricting direct access to an object's attributes, ensuring they are accessed or modified only through defined methods. These OOP principles have given me a solid foundation for creating scalable, efficient, and well-structured Python programs.

Day 3:

TASK :- Loops

The screenshot shows a Visual Studio Code editor with a Python file named `loops.py`. The code defines a list of names and iterates through them using a `for` loop. It also includes a `break` statement and a `continue` statement. The output in the terminal shows the names being printed.

```

loops.py
14 # print("Value is now equal to " + str(value))
15
16 names = ["Dave", "Sara", "John"]
17 # for x in names:
18 # print(x)
19
20 # for x in "Mississippi":
21 # print(x)
22
23 # for x in names:
24 # if x == "Sara":
25 # break
26 # print(x)
27
28 for x in names:
29     if x == "Sara":
30         continue
31     print(x)
32

```

Output in the terminal:

```

python-series/lesson08/loops.py
Dave
John

```

The screenshot shows a Visual Studio Code editor with a Java file named `strings`. The code calculates the factorial of 6 using a `while` loop. The output in the terminal shows the factorial calculation.

```

strings
1 i=9921004873-Gudala Guna Sankar reddy
2 fact=1
3 while(i<=6):
4     fact=fact*i
5     i=i+1
6
7 print("The factorial of 6 is:",fact)

```

Output in the terminal:

```

PS C:\Users\gunas\Desktop\javaproblems\strings> & C:/Python313/python.exe c:/Users/gunas/Desktop/javaproblems/strings/day3.py
Sorry, could not find the name
PS C:\Users\gunas\Desktop\javaproblems\strings> & C:/Python313/python.exe c:/Users/gunas/Desktop/javaproblems/strings/day3.py
The factorial of 6 is: 720
PS C:\Users\gunas\Desktop\javaproblems\strings>

```

Today, I explored essential control flow structures in programming, which are crucial for managing the logic and flow of a program. I began by learning about the **for loop**, a powerful tool for iterating over sequences like lists, strings, or ranges, enabling repetitive tasks to be performed efficiently. Following this, I studied the **while loop**, which continues executing a block of code as long as a

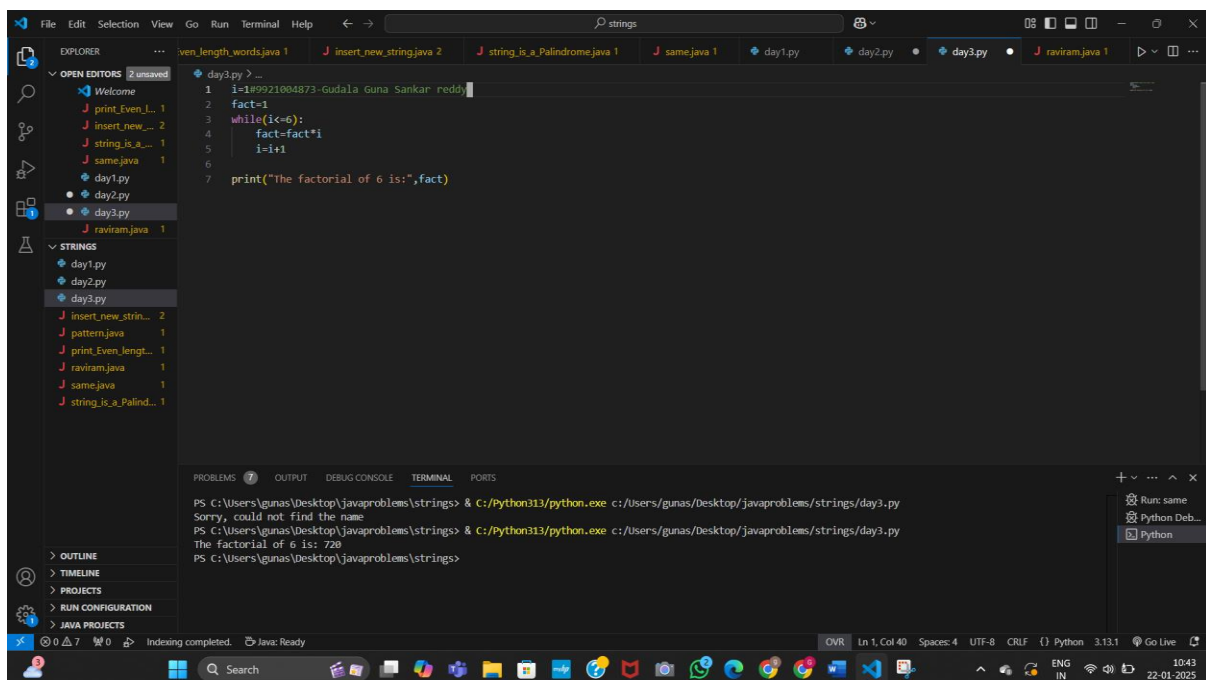
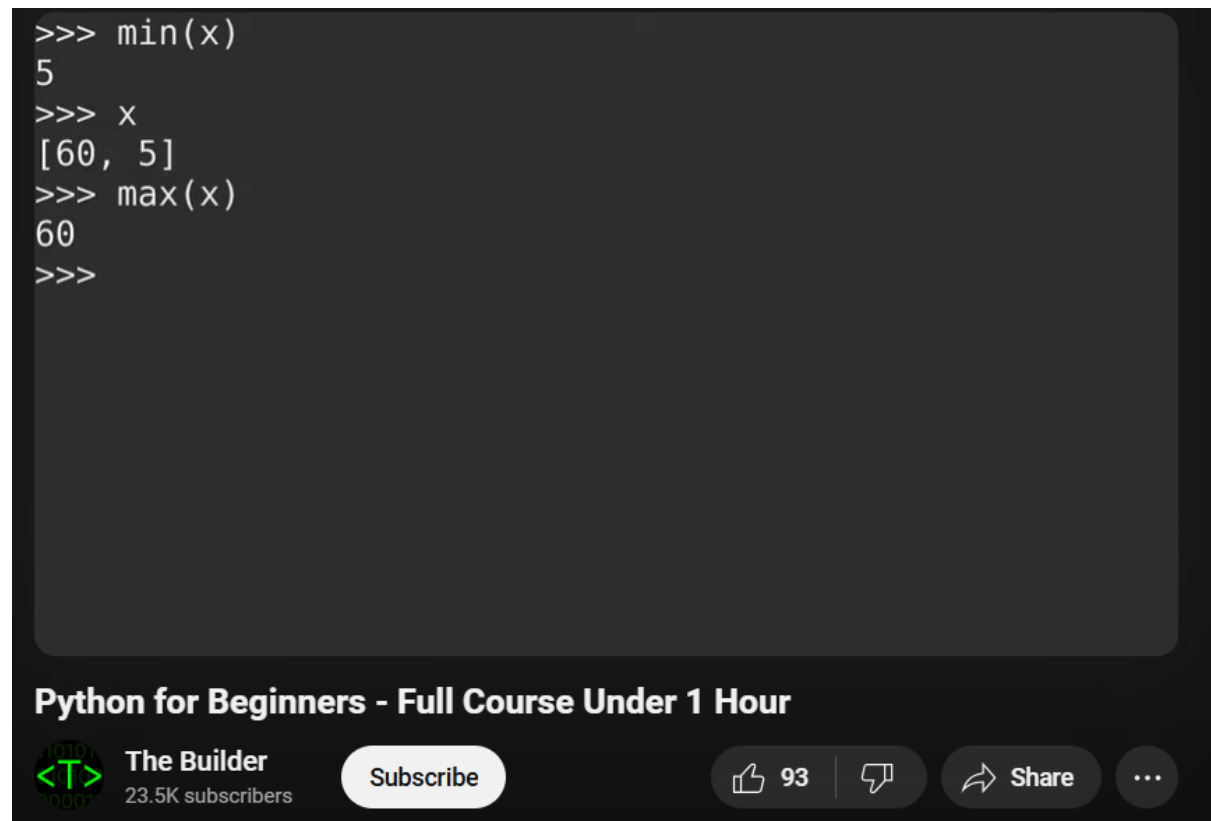
specified condition remains true, making it useful for scenarios where the number of iterations isn't predetermined.

Next, I explored the **do-while loop**, commonly found in languages like C and Java, where the code block executes at least once before the condition is evaluated, ensuring a minimum iteration. I then learned about the **switch statement**, a cleaner alternative to multiple if-else conditions, used to execute specific code blocks based on the value of a variable or expression, enhancing readability and structure.

Finally, I delved into the **break statement**, a control tool that exits a loop or a switch statement prematurely when a particular condition is met, providing a way to optimize code execution and prevent unnecessary iterations. These control flow constructs are fundamental for building dynamic and efficient programs.

Day4:

TASK :- More On Oops



Today, I deepened my understanding of object-oriented programming (OOP) concepts, which are essential for writing efficient and reusable code. I began by learning about **classes**, the blueprints for creating objects, and their ability to define properties and behaviors. I then explored **objects**, which

are instances of a class that hold specific data and utilize class-defined methods to perform actions. Next, I studied **methods**, which are functions within a class that define the actions an object can perform.

I also delved into **inheritance**, which allows a class (child class) to derive properties and methods from another class (parent class), fostering code reuse and extending functionality. Additionally, I explored **polymorphism**, the ability of different classes to be treated as instances of the same parent class, enabling a unified way to interact with objects while maintaining their specific behaviors.

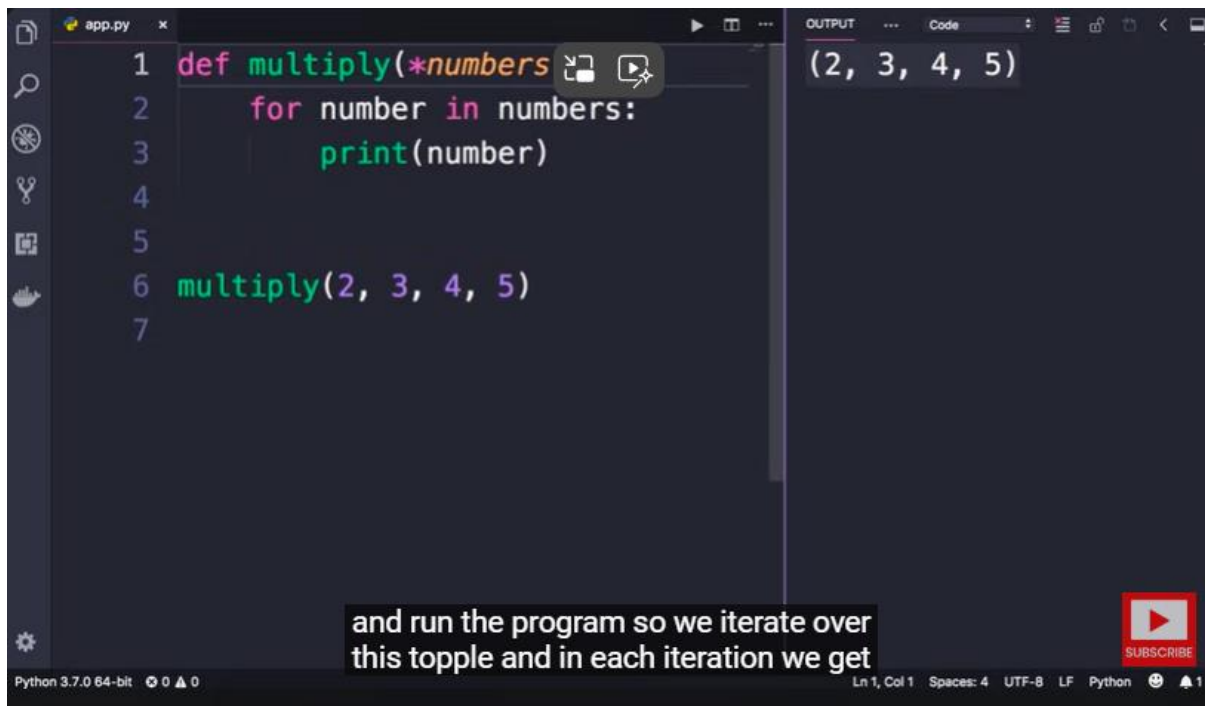
Data abstraction was another key concept, emphasizing the importance of exposing only necessary details while hiding complex implementation, making code simpler and more user-friendly. Lastly, I learned about **encapsulation**, which restricts direct access to an object's data by defining controlled methods for interacting with it, thereby enhancing security and data integrity.

I also studied the **types** of these principles:

1. **Inheritance Types:** Single, multiple, multilevel, hierarchical, and hybrid inheritance.
 2. **Polymorphism Types:** Compile-time (method overloading) and runtime (method overriding).
 3. **Encapsulation Types:** Public, private, and protected access modifiers.
- These concepts collectively strengthened my ability to design robust and scalable applications.

Day5:

TASK :- Functions Basic



The screenshot shows a Python IDE with a file named `app.py`. The code defines a function `multiply(*numbers)` that iterates over the `numbers` tuple and prints each number. The function is then called with `multiply(2, 3, 4, 5)`. The output window on the right shows the result: `(2, 3, 4, 5)`. A text overlay at the bottom reads: "and run the program so we iterate over this tuple and in each iteration we get".

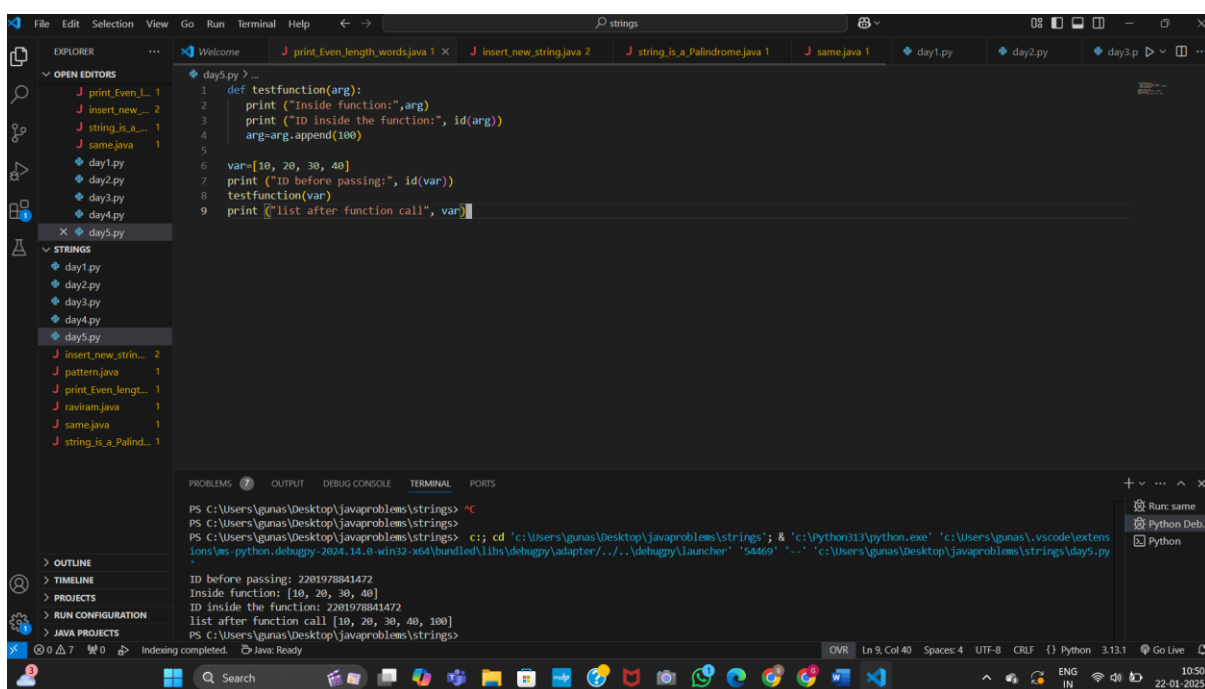
```

1 def multiply(*numbers):
2     for number in numbers:
3         print(number)
4
5
6 multiply(2, 3, 4, 5)
7

```

OUTPUT: (2, 3, 4, 5)

and run the program so we iterate over this tuple and in each iteration we get



The screenshot shows a Java IDE with a file named `day5.py`. The code defines a function `testfunction(arg)` that prints the ID of the argument, appends 100 to the list, and prints the ID of the modified list. The function is then called with `testfunction(var)`. The output window at the bottom shows the result: `ID before passing: 2201978841472`, `list after function call [10, 20, 30, 40, 100]`.

```

1 def testfunction(arg):
2     print("Inside function:", arg)
3     print("ID inside the function:", id(arg))
4     arg.append(100)
5
6 var=[10, 20, 30, 40]
7 print("ID before passing:", id(var))
8 testfunction(var)
9 print("list after function call", var)

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\gunas\Desktop\javaproblems\strings> ^C
PS C:\Users\gunas\Desktop\javaproblems\strings> ^C
PS C:\Users\gunas\Desktop\javaproblems\strings> c:; cd 'c:\Users\gunas\Desktop\javaproblems\strings'; & 'c:\python313\python.exe' 'c:\Users\gunas\.vscode\extensions\ms-python.debugpy-2024.14.0-win32-x64\bundled\libs\debugpy\adapter\..\..\debugpy\launcher' '54469' '-.' 'c:\Users\gunas\Desktop\javaproblems\strings\day5.py'

ID before passing: 2201978841472
Inside function: [10, 20, 30, 40]
ID inside the function: 2201978841472
list after function call [10, 20, 30, 40, 100]
PS C:\Users\gunas\Desktop\javaproblems\strings>

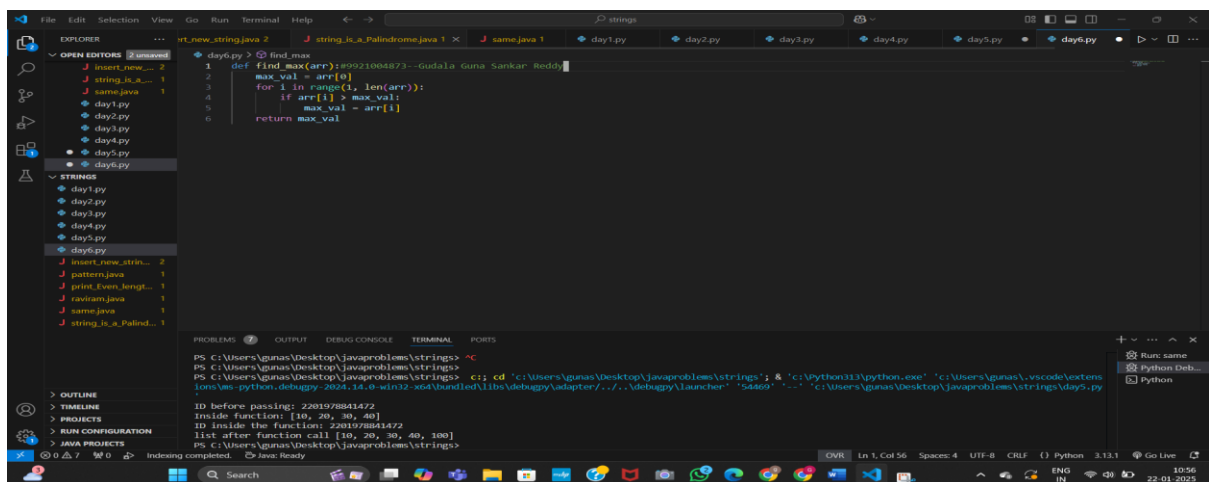
Today, I explored various types of functions in programming, which are fundamental for modular and reusable code. I began by studying **user-defined functions**, which are custom functions created by programmers to encapsulate specific logic or tasks. These functions are highly versatile and can accept parameters, return values, and be called multiple times within a program, making them essential for reducing redundancy and enhancing code readability.

Next, I learned about **lambda functions**, also known as anonymous functions. These are concise, single-expression functions that do not require a formal definition using the `def` keyword. Lambda functions are often used for short, throwaway operations and are particularly useful when passed as arguments to higher-order functions like `map()`, `filter()`, or `reduce()`.

Understanding these function types has provided me with a deeper appreciation of how to structure and optimize code, making it cleaner and more efficient.

Day6:

TASK :- PYTHON CODING(BIG Oh,space and time)



Space Complexity	Description	Examples
$O(1)$	Constant space	Finding max in an array
$O(n)$	Linear space	Storing a list of n elements
$O(n^2)$	Quadratic space	2D matrix operations
$O(\log n)$	Logarithmic space	Binary search recursion depth
$O(n \log n)$	Linearithmic space	Merge Sort
$O(2^n)$	Exponential space	Subset sum problem using recursion
$O(n!)$	Factorial space	Generating all permutations of a string

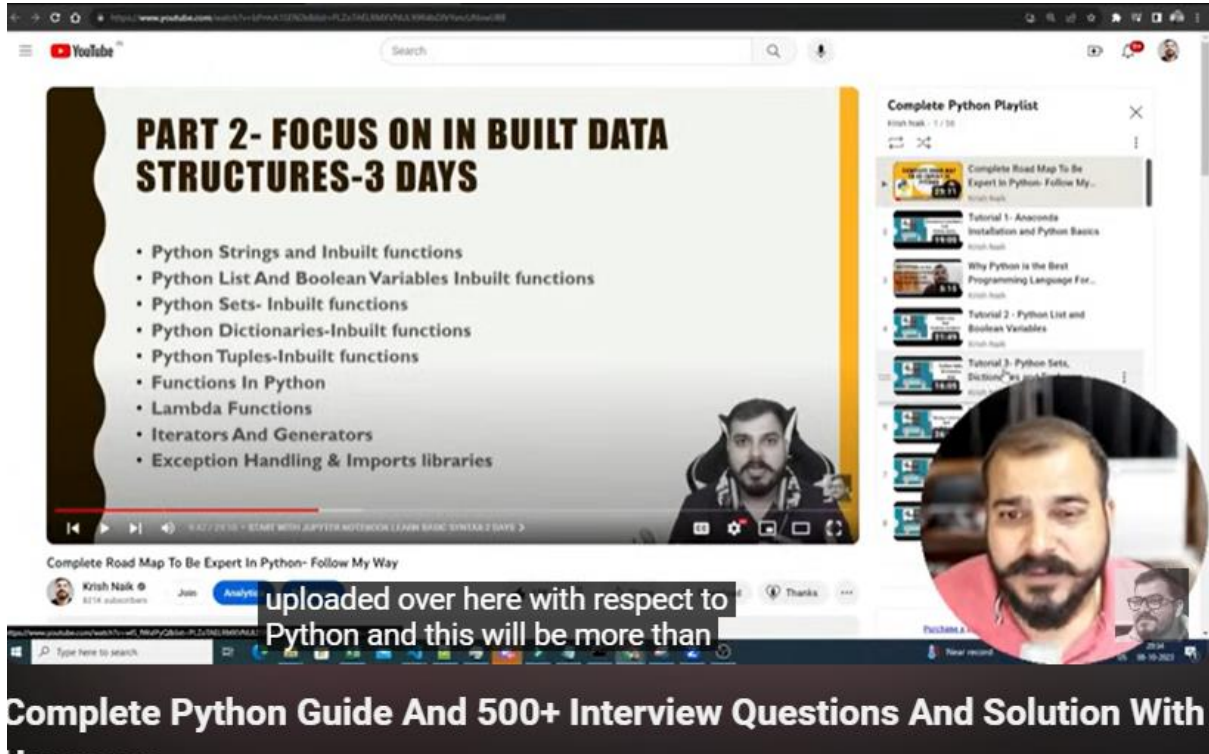
Today, I learned about **time and space complexity**, key concepts in analyzing the efficiency of algorithms. Time complexity measures the amount of time an algorithm takes to run as a function of its input size, while space complexity evaluates the amount of memory it requires during execution. These metrics are crucial for understanding the scalability and performance of algorithms.

I also explored **Big O notation**, a mathematical representation used to express the upper bound of an algorithm's growth rate. For instance, **$O(n)$** describes a linear time complexity, where the algorithm's runtime increases proportionally with the size of the input. Big O helps in comparing different approaches and choosing the most efficient one, particularly when working with large datasets.

Understanding these principles has enhanced my ability to evaluate and optimize algorithms, ensuring they perform efficiently in terms of both time and memory.

Day7:

TASK :- Simple Guide



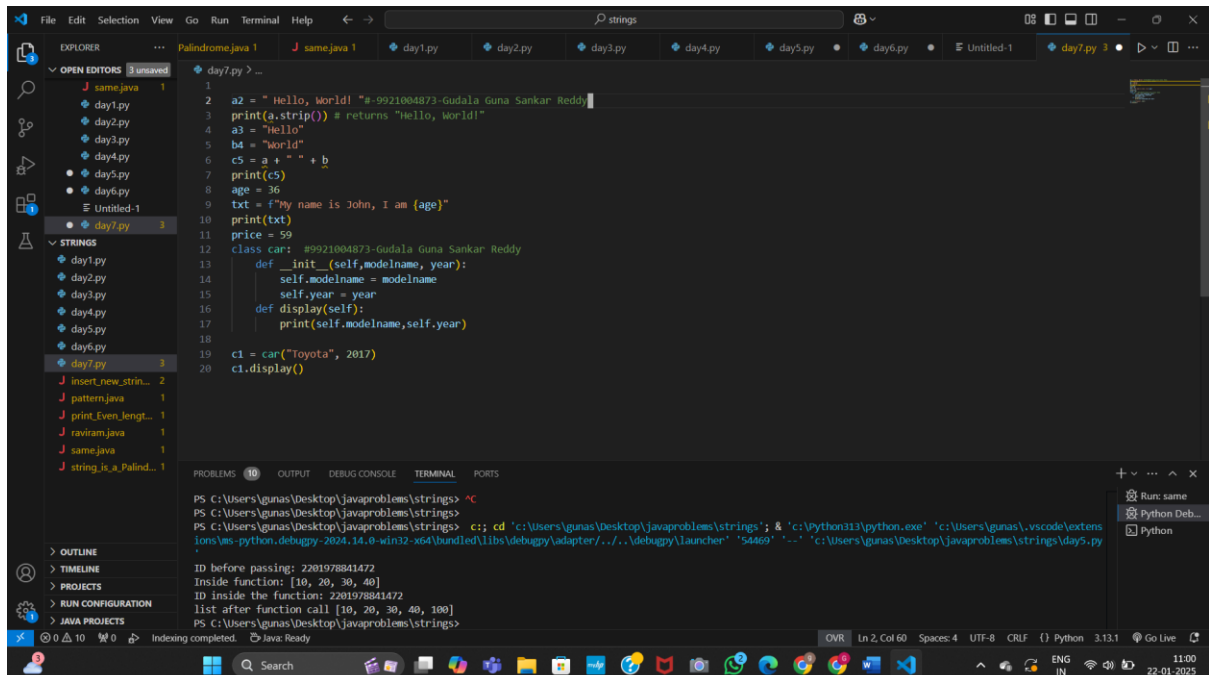
The image shows a YouTube video player interface. The video title is "PART 2- FOCUS ON IN BUILT DATA STRUCTURES-3 DAYS". The video content displays a list of topics to be covered:

- Python Strings and Inbuilt functions
- Python List And Boolean Variables Inbuilt functions
- Python Sets- Inbuilt functions
- Python Dictionaries-Inbuilt functions
- Python Tuples-Inbuilt functions
- Functions In Python
- Lambda Functions
- Iterators And Generators
- Exception Handling & Imports libraries

The video is by the channel "Krish Naik" and has 8274 subscribers. The video player shows a progress bar at 47:29/56. The video description mentions "Complete Road Map To Be Expert In Python- Follow My Way".

uploaded over here with respect to Python and this will be more than

Complete Python Guide And 500+ Interview Questions And Solution With



The image shows a screenshot of a code editor (VS Code) with the following code in the editor:

```
1 a2 = "Hello, World!" # 9921004873-Gudala Guna Sankar Reddy
2 print(a2.strip()) # returns "Hello, World!"
3 a3 = "Hello"
4 b4 = "World"
5 c5 = a + " " + b
6 print(c5)
7 age = 36
8 txt = f"My name is John, I am {age}"
9 print(txt)
10 price = 59
11 class car:
12     def __init__(self, modelname, year):
13         self.modelname = modelname
14         self.year = year
15         def display(self):
16             print(self.modelname, self.year)
17 c1 = car("Toyota", 2017)
18 c1.display()
```

The code editor also shows a file explorer on the left with files like "same.java", "day1.py", "day2.py", "day3.py", "day4.py", "day5.py", "day6.py", "day7.py", "insert_new_strin...", "pattern.java", "print_Even_lengt...", "raviram.java", "same.java", and "string_is_a_Palind...".

The terminal at the bottom shows the command prompt output:

```
PS C:\Users\gunas\Desktop\javaproblems\strings> ^C
PS C:\Users\gunas\Desktop\javaproblems\strings> c:\python313\python.exe c:\Users\gunas\vscode\extensions\ms-python.debugpy-2024.14.0-win32-x64\bundled\libs\debugpy\adapter\..\..\debugpy\launcher "5469" "..." c:\Users\gunas\Desktop\javaproblems\strings\day5.py
ID before passing: 2201978841472
Inside function: [10, 20, 30, 40]
ID inside the function: 2201978841472
list after function call [10, 20, 30, 40, 100]
PS C:\Users\gunas\Desktop\javaproblems\strings>
```

Today, I consolidated my understanding of **all the core concepts of Python**, building a strong foundation for versatile programming. I revisited essential topics such as variables, data types, control structures, functions, object-oriented programming principles, and advanced concepts like decorators, generators, and error handling. This comprehensive review reinforced my ability to write efficient and scalable Python code.

Additionally, I focused on applying my knowledge by solving as many **practice problems** as possible. These problems helped me improve my problem-solving skills, enhance logical thinking, and deepen my grasp of Python's functionality. Through this hands-on practice, I gained confidence in tackling real-world challenges and writing clean, optimized, and effective solutions.