THE ROLE OF VARIABLES, DATA TYPES, INPUT AND OUTPUT FUNCTIONS, AND CONDITIONAL STATEMENTS IN CREATING INTERACTIVE PROGRAMS.

DATA DYNAMOS

DUNCAN NUKERI
SIPHIWE MKHWANAZI
NOZIPO NTIYO
LERATO MOTAUNG
KHENSANI KUBAYI
SHAMAINE MUKHITHI

VARIABLES

DATA TYPES

INPUT AND OUTPUT FUNCTIONS

CONDITIONAL STATEMENTS

VARIABLES

WHAT IS A VARIABLE ?

- Variables are a fundamental concept in computer programming, serving as the basic building blocks for storing and manipulating data.
- Based on the data type of a variable, the computer's memory allocates space and determines what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimal values, or characters in these variables.

RULES FOR PYTHON VARIABLES

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number.
- A variable name can only contain alpha-numeric characters and underscores (A-Z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)
- Spaces are not allowed in variables, but underscore can be used to separate in variable names.

ASSIGNING VALUES TO VARIABLES

In Python, variables do not require explicit declaration to allocate memory. Memory reservation occurs automatically when you assign a value to a variable. The equal sign serves as the assignment operator to give values to variables.

The operand to the left of the = operator is the name of the variable and the operand to the right of the = operator is the value stored in the variable.

FOR EXAMPLE

```
a= 25          # An integer assignment
b = 5000.0      # A floating point
c = "Mark"       # A string
print(a)
print (b)
print (c)
```

✓ 0.0s

```
25
5000.0
Mark
```

Local Variable

A variable declared within a function is called local variable. It is not callable outside the function.

```
    # def myfunc:
    #A local variable
    a = 30
    print(a)
```

✓ 0.0s

```
30
```

Global Variable

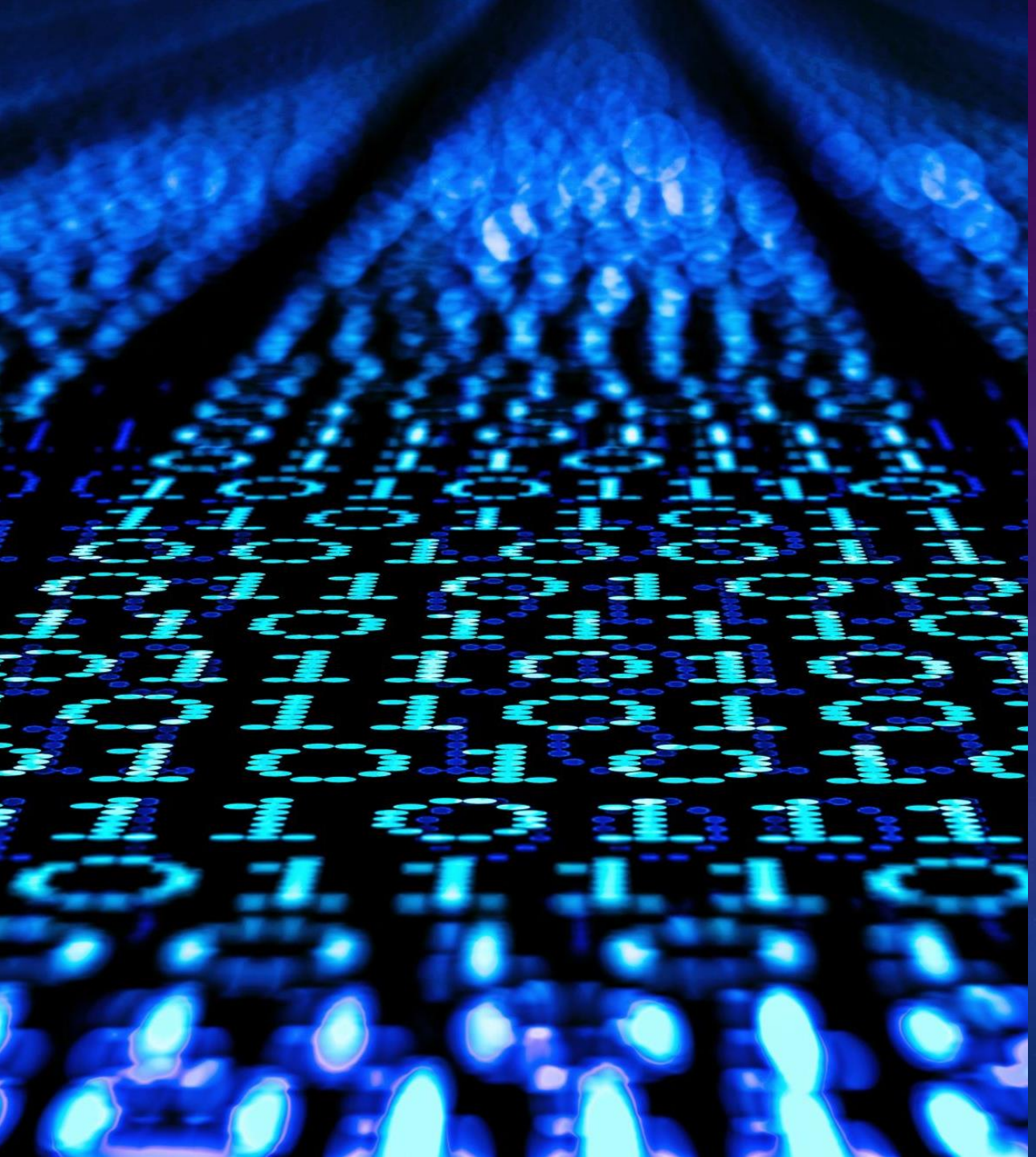
A variable created in Python is usually a global variable, except for the case of local variable. It is callable anywhere in Python

```
    # def myfunc:
    #A local variable
    x = 5
    print(x)

    #A global variable
    y = 10
    print(y)
```

✓ 0.0s

```
5
10
```



CONCLUSION

A variable gives a name to a value

- Choose a meaningful name

Reading and Assignment

- A variable can be read in an expression
- A variable can be changed in an assignment statement

The use of variables

- To replace a complex expression with several simpler ones
- To hold an ‘input’ string from the user.

DATA TYPES

WHAT ARE DATA TYPES?

Data types are fundamental in programming, serving as the building blocks for data management and interaction.

PYTHON HAS THE FOLLOWING DATA TYPES BUILT-IN BY DEFAULT, IN THESE CATEGORIES:

Text Type	str
Numeric Types	Int, float, complex
Sequence Types	List, tuple, range
Mapping Types	dict
Set Types	Set, frozenset
Boolean Type	bool
Binary Types	Bytes, bytearray, memoryview
None Type	NoneType

IN PYTHON, UNDERSTANDING DATA TYPES—BOTH PRIMITIVE AND COMPOSITE—ENHANCES OUR ABILITY TO CREATE INTERACTIVE APPLICATIONS THAT RESPOND EFFECTIVELY TO USER INPUT.

PRIMITIVE DATA TYPES

- Integers (int): Represent whole numbers, useful for counting, indexing, or setting numeric parameters.
- Floating-point numbers (float): Used for decimals, important in calculations involving currency or measurements.
- Strings (str): Handle text data, critical for user inputs like names, messages, or commands.
- Booleans (bool): Represent truth values (True or False), essential for decision-making processes.

COMPOSITE DATA TYPES

- Lists: Store collections of items, allowing dynamic user inputs like selections or responses.
- Dictionaries: Key-value pairs used to manage data that requires quick lookups, such as user settings.
- Tuples and Sets: Useful for fixed collections and unique items, respectively.

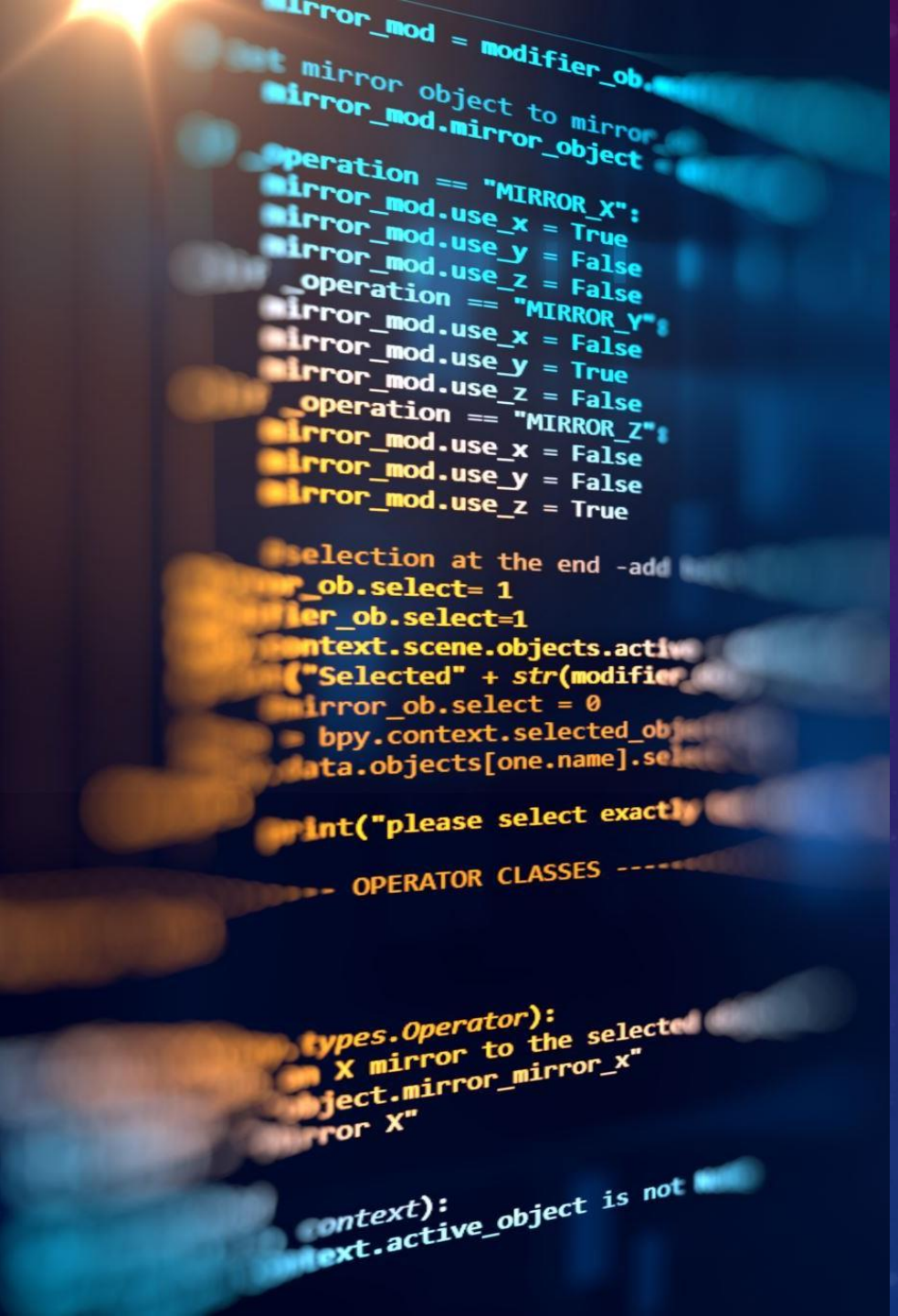
Enhancing User Interaction

Data types facilitate user interaction in several keyways:

Input Validation: Ensures that user inputs match expected types, minimizing errors. For instance, asking for a user's age and validating it as an integer prevents crashes and incorrect data processing.

Data Manipulation: Different data types allow various operations. For example, lists support methods like append() and sort(), enabling dynamic changes based on user input. This flexibility is vital for applications that need to process user choices in real time.

Control Flow: C to execute different code paths. A boolean flag (e.g., is_logged_in) can determine access to certain features, tailoring the experience based on user status.



ERROR HANDLING AND FEEDBACK

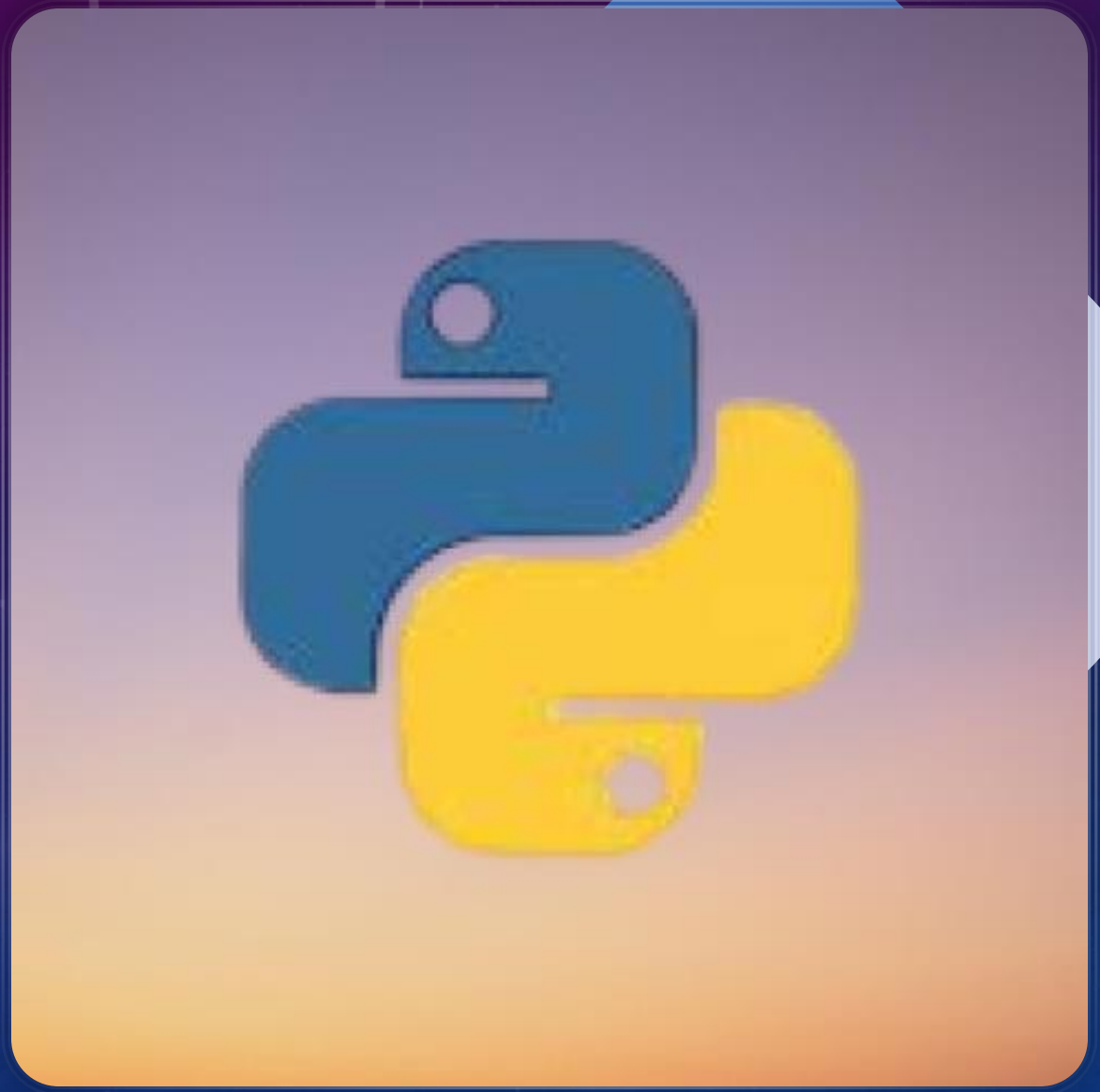
EFFECTIVE ERROR HANDLING AND USER FEEDBACK ARE CRITICAL IN INTERACTIVE PROGRAMS

- **Error Handling**: By using try and except blocks, developers can gracefully manage type-related errors. For instance, if a user inputs a string when a number is expected, the program can catch the exception and prompt the user to enter a valid number, enhancing the user experience.
- **User Feedback:** The clear presentation of information is crucial. Using formatted strings allows programs to communicate effectively with users. For example, displaying scores or statuses in a user-friendly manner improves clarity and engagement. .

Conclusion

- In summary, data types play a vital role in developing interactive programs. They facilitate input validation, enable data manipulation, and govern control flow, all of which are essential for creating responsive applications. Furthermore, robust error handling and clear user feedback enhance usability, ensuring that programs can interact effectively with users.
- Understanding how to leverage data types not only leads to smoother functionality but also enriches the overall user experience, making your applications more intuitive and reliable. As you develop interactive programs, always consider the role of data types in shaping how users interact with your application.

INPUT FUNCTIONS



WHAT IS INPUT FUNCTION

User input is a fundamental concept the allows developers to create interactive and dynamic applications. You can define the input function using the built-in input() function which will allow developers to get user input from the console. This function prompts the user to enter data, which is then captured and stored as a string.

THE IMPORTANCE OF USER INPUT

User input is a crucial aspect of software development as it allows programs to adapt to the needs and preferences of the end-user. By accepting input from the user, applications can become more personalized, responsive, and tailored to the user's specific requirements.

```
user_input = input ("Please enter your name: ")
print ("Helllo, " + user_input + "!")
```

```
Please enter your name:  Data Dynamos
Helllo, Data Dynamos!
```

Practical Examples and Application of User Input

- Data entry
- Configuration settings
- Menu driven application

HANDLING DIFFERENT DATA TYPES

By default, the input() function returns the user's input as a string. However, in many cases, you may need to convert the input to a different data type, such as an integer, boolean, float, string. Python provides several built-in functions for this purpose:

- int(user_input)
- float(user_input)
- bool(user_input)
- str(user_input)

```
user_input1 = input ("Please enter your name: ")
print (" Hello, " + user_input1 + "!")
print(type(user_input1))
#next line
user_input2 = int(input ("Please enter your age: "))
print (f'You are  + {user_input2} + years old' )
print(type(user_input2))
```

```
Please enter your name:  Data Dynamos
 Hello, Data Dynamos!
<class 'str'>
Please enter your age:  20
You are  + 20 + years old
<class 'int'>
```

HANDLING INVALID INPUTS

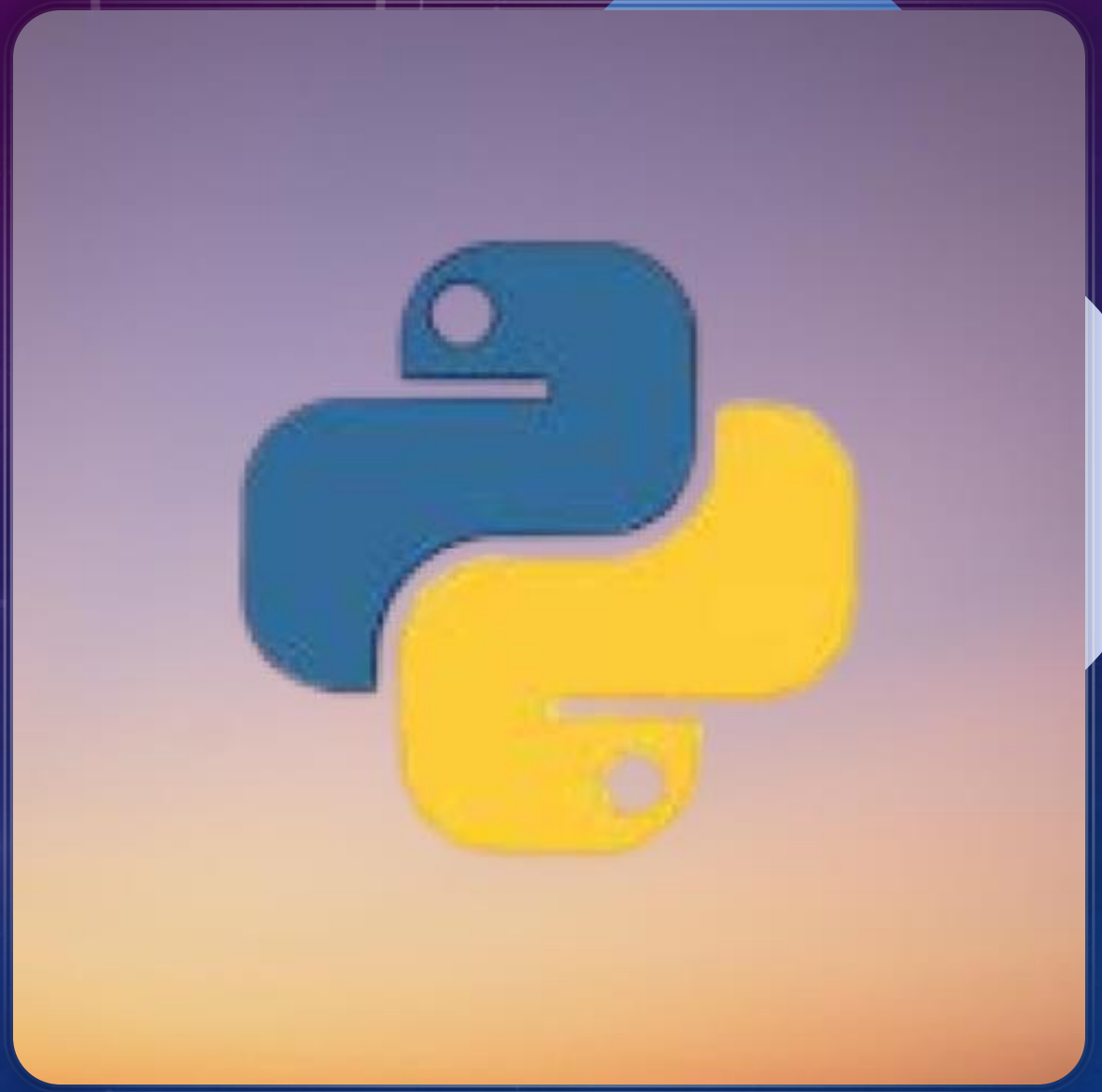
When working with user input, it's important to consider the possibility of invalid input. You can use exception handling techniques to handle these cases and provide appropriate error messages.

```python
try:
    age = int(input("Please enter your age: "))
    if age >= 18:print("You are an adult.")
    else:
        print("You are a minor.")
except ValueError:
    print("Invalid input. Please enter a valid integer.")
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  JUPYTER

PS C:\Users\Shamaine.Mukhithi> & C:/Python311/python.exe c:/Users/Shamaine.
Please enter your age: twenty
Invalid input. Please enter a valid integer.
PS C:\Users\Shamaine.Mukhithi> 20
20
PS C:\Users\Shamaine.Mukhithi>
```

OUTPUT FUNCTIONS



WHAT IS OUTPUT FUNCTIONS

Output functions in creating interactive programs are methods that display or communicate information to the user.

OUTPUT FUNCTIONS PLAY A CRUCIAL ROLE IN CREATING INTERACTIVE PROGRAMS BY:

- **Displaying results: Showing calculated or processed data.**
- **Providing feedback: Informing users about input validity or errors.**
- **Guiding user interaction: Prompts, menus, and instructions.**
- **Visualizing data: Charts, graphs, and other visual representations.**
- **Enhancing user experience: Animations, transitions, and effects.**

PRINT ()

IT IS USED TO DISPLAY THE OUTPUT ON THE CONSOLE

Text in double quotes will consider as Normal text and display on output screen

```
name = "Data Dynamos"
age = 25
print ("Name: ", name, "Age: ", age)
```

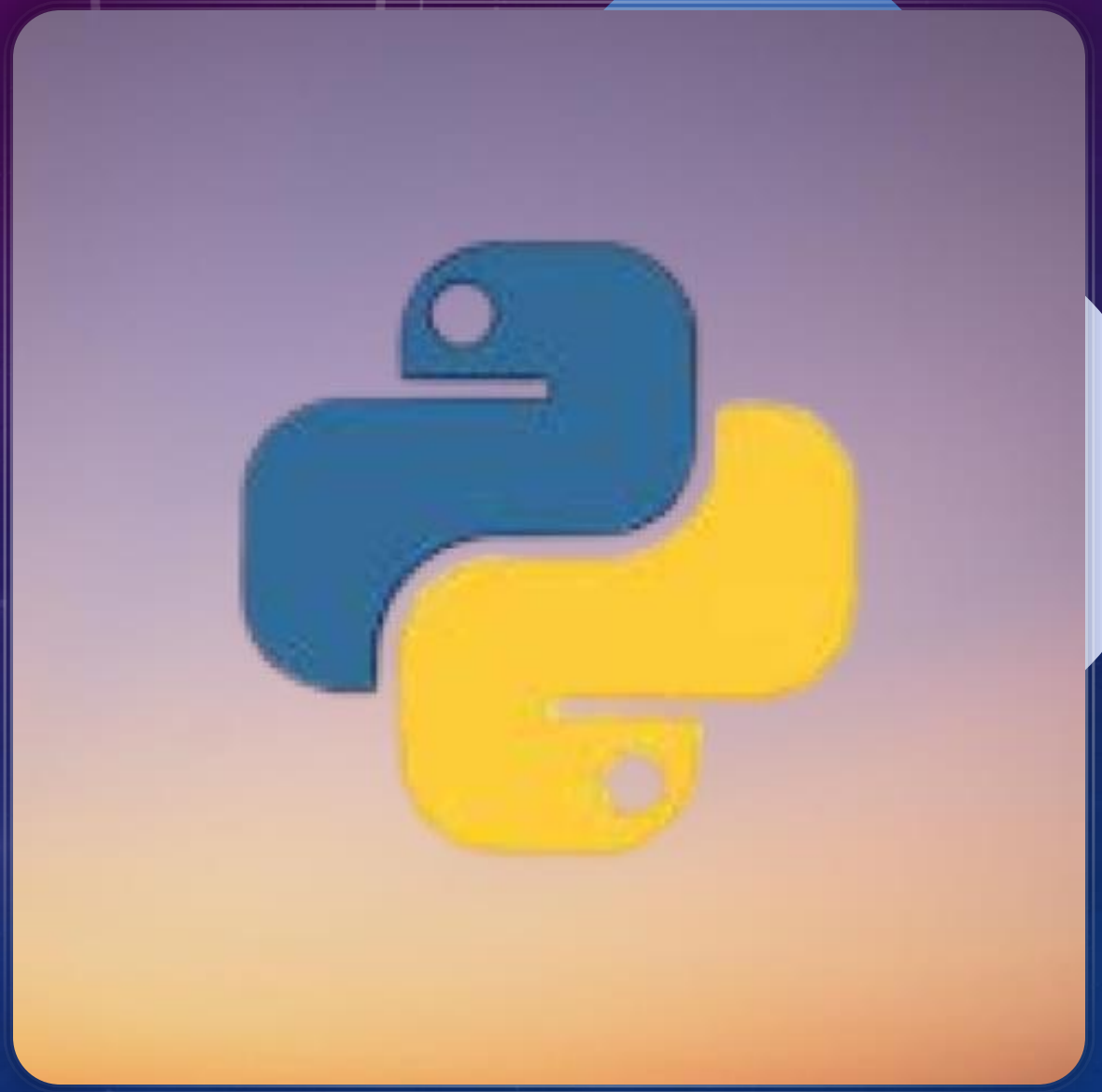
```
Name:  Data Dynamos Age:  25
```

In this code , we are passing two parameters name and age to the print function

```
name = "Data Dynamos"
age = 25
print ("Hello my name is:", name, "and I am", age, "years old.")
```

```
Hello my name is: Data Dynamos and I am 25 years old.
```

CONDITIONAL STATEMENTS



WHAT ARE CONDITIONAL STATEMENTS

Conditional statements in Python are used to control the flow of a program based on conditions or decisions. They allow your code to execute different actions depending on specific conditions.

CONDITIONAL OPERATORS

==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

LOGICAL OPERATORS

- **And** (&&) **(Both conditions must be true)**

- We use the And keyword as logical operator to combine conditional statements: Test if a is greater than b**, AND** If c is greater than a.

- **Or** (/) **(At least one condition must be true)**

- The Or keyword is also a logical operator that is used to combine conditional statements: Test if a is greater than b, **OR** if a is greater than c.

- **Not** (!) **(Negates the condition)**

- The Not keyword is a logical operator, and is used to reverse the reverse of the conditional statement: Test if a **NOT** greater than b.

CONDITIONAL STATEMENTS

If Statement

An If statement is used to execute a code if the condition is true.

```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

If-Elif Statement

The Elif keyword is a way of saying "if the previous conditions were not true, then try this condition.

```
a = 33
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
```

CONDITIONAL STATEMENT

If Else Statement

The else statement catches anything which isn't caught by the preceding conditions.

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

Short Hand If

If you only have one statement to execute, you can put it on the same line as the if statement.

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

CONDITIONAL STATEMENTS

Short Hand If...Else

If you have only one statement to execute, one for if, and one for else, you can put it all on the same line.

```
a = 2

b = 330

print("A") if a > b else print("B")
```

The pass statement

If statements cannot be empty, but if you for some reason have an If statement with no content, put in the pass statement to avoid getting an error.

```
a = 33
b = 200

if b > a:
  pass
```

THANK YOU