Subscribe for PMS, Maths, SA Lab, DCET videos
https://www.youtube.com/@RaviRnandi
Videos by : Ravi kumar R,Lecturer in science,GPT Bagepalli

UNIT - 4
Introduction to Python Programming:

Python is a high-level, interpreted, and versatile programming language known for its simplicity, readability, and wide range of applications.

Guido van Rossum created Python in the late 1980s, and it was first released in **1991.**

Python's design philosophy emphasizes code readability and a clean syntax, making it easy for beginners to learn and use effectively.

Key Features of Python:

1. **Easy to Learn**: Python's syntax is simple and straightforward, making it an ideal language for beginners to start their programming journey.
2. **Highly Readable**: The use of indentation for code blocks and the absence of unnecessary symbols and keywords make Python code highly readable.
3. **Interpreted Language**: Python code is executed line by line, allowing for quick prototyping and ease of debugging.
4. **Dynamic Typing**: Python is dynamically typed, meaning you don't need to declare variable types explicitly.
5. **Extensive Libraries**: Python comes with a vast collection of standard libraries and third-party modules, providing solutions for various tasks.
6. **Cross-platform**: Python code can run on different platforms without modification, making it highly portable.

Example 1: Hello, World!

Let's start with the traditional "Hello, World!" program in Python:

```
python                                                    Copy code

print("Hello, World!")
```

```
print("Hello, World!")
```

Syntax in Programming:
In programming, "syntax" refers to the **set of rules that dictate how code should be written to be considered valid and executable by a computer**. Just like human languages have grammar rules, programming languages have syntax rules. A single mistake in the syntax can lead to errors and prevent the code from running correctly.

Syntax of Python:
The syntax of Python refers to the specific rules and guidelines that must be followed when writing Python code. Python's syntax is designed to be easy to read and understand, which makes it popular among beginners and experienced developers alike.

Example 2: Variables and Basic Arithmetic

Python allows you to create variables without specifying their data types:

python Copy code

```python
x = 10
y = 5

sum = x + y
difference = x - y
product = x * y
quotient = x / y

print("Sum:", sum)
print("Difference:", difference)
print("Product:", product)
print("Quotient:", quotient)
```

```
x = 10
y = 5

sum = x + y
difference = x - y
product = x * y
quotient = x / y

print("Sum:", sum)
```

```
print("Difference:", difference)
print("Product:", product)
print("Quotient:", quotient)
```

Example 3: Conditional Statements

Python supports if-else statements for decision-making:

python Copy code

```python
age = 18

if age >= 18:
    print("You are an adult.")
else:
    print("You are a minor.")
```

```
age = 18

if age >= 18:
   print("You are an adult.")
else:
   print("You are a minor.")
```

Example 4: Loops

Python provides for and while loops for iteration:

python Copy code

```python
# For loop to print numbers from 1 to 5
for i in range(1, 6):
    print(i)

# While loop to print numbers from 1 to 5
i = 1
while i <= 5:
    print(i)
    i += 1
```

```
# For loop to print numbers from 1 to 5
```

```
for i in range(1, 6):
    print(i)

# While loop to print numbers from 1 to 5
i = 1
while i <= 5:
    print(i)
    i += 1
```

Example 5: Functions

Functions allow you to define reusable blocks of code:

python Copy code

```
def greet(name):
    print("Hello, " + name + "!")

greet("Alice")
greet("Bob")
```

```
def greet(name):
    print("Hello, " + name + "!")

greet("Alice")
greet("Bob")
```

Comments in Python:
In Python, comments are non-executable lines of text that are used to add explanations or notes within the code. Comments are intended for developers to understand the code better and are ignored by the Python interpreter during execution. They play a crucial role in making the code more readable and self-explanatory.

Types of Comments in Python:
Single-line Comments: These comments start with the # symbol and extend only to the end of the line. Anything written after the # symbol on the same line is considered a comment and is not executed.

python Copy code

```python
# This is a single-line comment in Python.
print("Hello, World!")  # This is also a comment.
```

This is a single-line comment in Python.
print("Hello, World!") # This is also a comment.

1. Multi-line Comments (Docstrings): Multi-line comments are created using triple quotes ''' or """. They are used for multi-line explanations and are often referred to as "docstrings" when used for documenting functions or modules.

python Copy code

```python
'''
This is a multi-line comment.
It can span multiple lines.
'''
print("Hello, World!")

"""
This is another multi-line comment.
It can also span multiple lines.
"""
print("Hello, Python!")
```

'''
This is a multi-line comment.
It can span multiple lines.
'''
print("Hello, World!")

"""
This is another multi-line comment.
It can also span multiple lines.
"""
print("Hello, Python!")

Example Program with Comments:

python Copy code

```python
# This is a simple Python program to calculate the area of a rectangle.

# Function to calculate the area of a rectangle
def calculate_area(length, width):
    # This is a comment inside the function.
    area = length * width
    return area

# Input values
length = 5
width = 3

# Calculate and print the area
result = calculate_area(length, width)
print("The area of the rectangle is:", result)

# The program ends here.
```


```
# This is a simple Python program to calculate the area of a rectangle.
# Function to calculate the area of a rectangle
def calculate_area(length, width):
    # This is a comment inside the function.
    area = length * width
    return area

# Input values
length = 5
width = 3

# Calculate and print the area
result = calculate_area(length, width)
print("The area of the rectangle is:", result)
```

```
# The program ends here.
```

In the above program, comments are used to explain the purpose of the code, the function's working, and the input values. When the program is executed, all the comments are ignored, and only the executable code is executed. Comments are an essential part of programming as they improve code readability, help other developers understand the code, and make code maintenance easier.

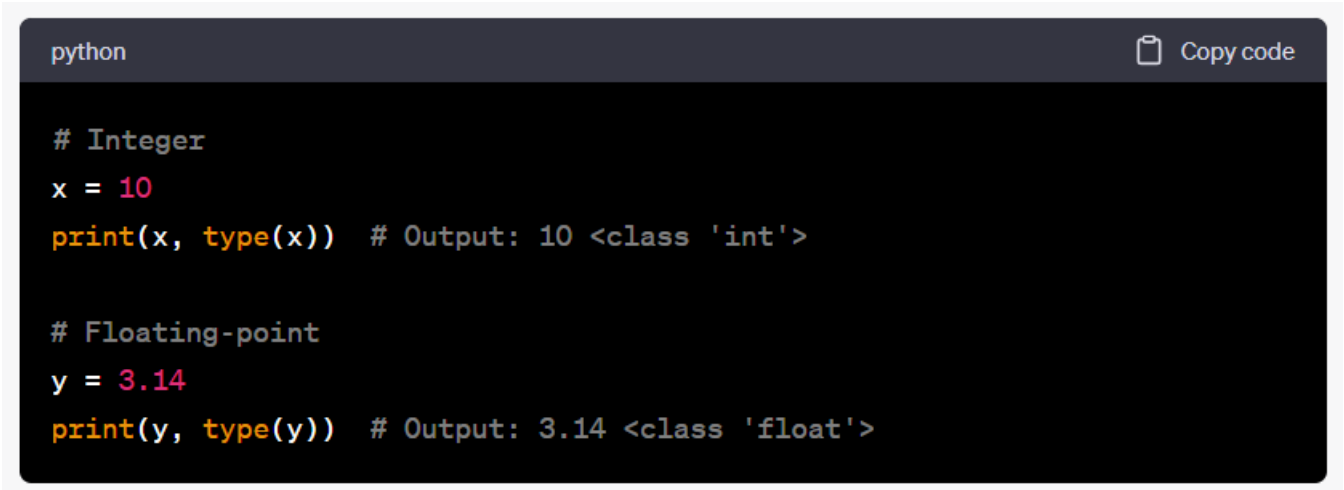
Data Types in Python:

In Python, data types represent the type of data that can be stored in a variable. Each data type has specific characteristics and operations associated with it. Python is a dynamically-typed language, meaning you don't need to explicitly declare the data type of a variable; Python infers it based on the value assigned to the variable.

Common Data Types in Python:

1. Numeric Types:
    1. int: Represents integer numbers (whole numbers without a fractional part). Example: x = 10

    2. float: Represents floating-point numbers (numbers with a decimal point). Example: y = 3.14




```
# Integer
x = 10
print(x, type(x))  # Output: 10 <class 'int'>

# Floating-point
y = 3.14
print(y, type(y))  # Output: 3.14 <class 'float'>
```

2. String:

str: Represents a sequence of characters enclosed within single (' ') or double (" ") quotes. Example: name = "John"

```
name = "Alice"
print(name, type(name))
# Output: Alice <class 'str'>
```

3. Boolean:
   bool: Represents the Boolean values True or False (used for logical operations). Example: is_student = True

3. Boolean:

python Copy code

```python
is_student = True
print(is_student, type(is_student))  # Output: True <class 'bool'>
```

```
is_student = True
print(is_student, type(is_student))
 # Output: True <class 'bool'>
```

4. Sequence Types:
   1. list: Represents an ordered collection of elements, enclosed within square brackets [ ]. Example: my_list = [1, 2, 3, 4, 5]

   2. tuple: Represents an ordered, immutable collection of elements, enclosed within parentheses ( ). Example: my_tuple = (10, 20, 30)

4. Sequence Types:

python Copy code

```python
# List
numbers = [1, 2, 3, 4, 5]
print(numbers, type(numbers))  # Output: [1, 2, 3, 4, 5] <class 'list'>

# Tuple
coordinates = (10, 20)
print(coordinates, type(coordinates))  # Output: (10, 20) <class 'tuple'>
```

```
# List
numbers = [1, 2, 3, 4, 5]
print(numbers, type(numbers))
# Output: [1, 2, 3, 4, 5] <class 'list'>

# Tuple
coordinates = (10, 20)
print(coordinates, type(coordinates))
# Output: (10, 20) <class 'tuple'>
```

5. Mapping Type: dict: Represents a collection of key-value pairs, enclosed within curly braces { }. Example: my_dict = {'name': 'John', 'age': 25}

5. Mapping Type:

python Copy code

```
person = {'name': 'John', 'age': 30, 'city': 'New York'}
print(person, type(person))  # Output: {'name': 'John', 'age': 30, 'city': '
```

```
person = {'name': 'John', 'age': 30, 'city': 'New York'}
print(person, type(person))

# Output: {'name': 'John', 'age': 30, 'city': 'New York'} <class 'dict'>
```

6. Set Types:
   1. set: Represents an unordered collection of unique elements, enclosed within
   curly braces { }. Example: my_set = {1, 2, 3, 4, 5}
   2. frozenset: Represents an immutable version of a set. Example: my_frozenset = frozenset({1, 2, 3})

6. Set Types:

```python
# Set
unique_numbers = {1, 2, 3, 4, 5}
print(unique_numbers, type(unique_numbers))  # Output: {1, 2, 3, 4, 5} <clas

# Frozenset
immutable_numbers = frozenset({1, 2, 3})
print(immutable_numbers, type(immutable_numbers))  # Output: frozenset({1, 2
```


```
# Set
unique_numbers = {1, 2, 3, 4, 5}
print(unique_numbers, type(unique_numbers))

# Output: {1, 2, 3, 4, 5} <class 'set'>

# Frozenset
immutable_numbers = frozenset({1, 2, 3})
print(immutable_numbers, type(immutable_numbers))

 # Output: frozenset({1, 2, 3}) <class 'frozenset'>
```


7. None Type:

None: Represents a special data type that denotes the absence of a value or a null value. Example: result = None

7. None Type:

```python
result = None
print(result, type(result))  # Output: None <class 'NoneType'>
```


```
result = None
print(result, type(result))  # Output: None <class 'NoneType'>
```

Introduction to If-Else Statements in Python

1. **Introduction to Conditional Statements:** Conditional statements in programming allow us to make decisions based on certain conditions. One of the most fundamental forms of a conditional statement is the if-else statement. In Python, the if-else statement is used to execute different blocks of code based on whether a given condition is True or False.
2. **Syntax of the If-Else Statement:**
   The general syntax of the if-else statement in Python is as follows:

```python
if condition:
    # Code to be executed if the condition is True
else:
    # Code to be executed if the condition is False
```


```
if condition:
    # Code to be executed if the condition is True
else:
    # Code to be executed if the condition is False
```


3. Example Program: Let's look at a simple example to understand how if-else statements work:

```python
# Program to check if a number is even or odd

# Input
num = int(input("Enter a number: "))

# Condition check using if-else
if num % 2 == 0:
    print(num, "is even.")
else:
    print(num, "is odd.")
```


```
# Program to check if a number is even or odd

# Input
num = int(input("Enter a number: "))
```

```
# Condition check using if-else
if num % 2 == 0:
   print(num, "is even.")
else:
   print(num, "is odd.")
```

4. Explanation of the Program:

We start by taking user input using the input() function and converting it to an integer using int().

The if statement checks if the remainder of num divided by 2 is equal to 0. If this condition is True, the code inside the if block is executed, which prints that the number is even.

If the condition is False, the code inside the else block is executed, which prints that the number is odd.

5. Nested If-Else Statements: You can also nest if-else statements inside each other to handle more complex conditions. The inner if-else statements are indented further.

6. Multiple Conditions: You can use multiple elif (short for "else if") statements to handle multiple conditions. The program will execute the block of code associated with the first True condition.

7. Summary: if-else statements are crucial in programming to implement decision-making logic. They allow us to create programs that respond dynamically to different scenarios based on conditions. By combining if and else statements, we can control the flow of our code and make it more versatile and adaptable.

Example 1: Checking if a Number is Positive or Negative

python Copy code

```python
# Input
num = int(input("Enter a number: "))

# Checking if the number is positive or negative
if num >= 0:
    print("The number is positive.")
else:
    print("The number is negative.")
```

```
# Input
```

```
num = int(input("Enter a number: "))

# Checking if the number is positive or negative
if num >= 0:
   print("The number is positive.")
else:
   print("The number is negative.")
```

Example 2: Finding the Largest of Two Numbers

python Copy code

```python
# Input
num1 = int(input("Enter the first number: "))
num2 = int(input("Enter the second number: "))

# Comparing and finding the largest number
if num1 > num2:
    print(num1, "is larger.")
else:
    print(num2, "is larger.")
```

```
# Input
num1 = int(input("Enter the first number: "))
num2 = int(input("Enter the second number: "))

# Comparing and finding the largest number
if num1 > num2:
   print(num1, "is larger.")
else:
   print(num2, "is larger.")
```

Example 3: Determining Pass/Fail based on Exam Score

python Copy code

```python
# Input
score = int(input("Enter your exam score: "))

# Checking if the student passed or failed
if score >= 50:
    print("Congratulations! You passed the exam.")
else:
    print("Sorry, you failed the exam. Keep studying!")
```


```
# Input
score = int(input("Enter your exam score: "))

# Checking if the student passed or failed
if score >= 50:
   print("Congratulations! You passed the exam.")
else:
   print("Sorry, you failed the exam. Keep studying!")
```


Example 4: Categorizing Age Groups

python Copy code

```python
# Input
age = int(input("Enter your age: "))

# Categorizing age groups
if age < 18:
    print("You are a minor.")
elif age >= 18 and age < 60:
    print("You are an adult.")
else:
    print("You are a senior citizen.")
```


```
# Input
age = int(input("Enter your age: "))
```

```
# Categorizing age groups
if age < 18:
   print("You are a minor.")
elif age >= 18 and age < 60:
   print("You are an adult.")
else:
   print("You are a senior citizen.")
```

Example 5: Determining Leap Year

python Copy code

```
# Input
year = int(input("Enter a year: "))

# Checking if the year is a leap year or not
if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
    print(year, "is a leap year.")
else:
    print(year, "is not a leap year.")
```

```
# Input
year = int(input("Enter a year: "))

# Checking if the year is a leap year or not
if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
   print(year, "is a leap year.")
else:
   print(year, "is not a leap year.")
```

Title: Introduction to Loops in Python

Notes:

1. Introduction to Loops: Loops are an essential concept in programming that allow us to execute a block of code repeatedly. Python provides two main types of loops: for and while. Loops are incredibly useful for automating repetitive tasks and processing collections of data.

2. for Loop: The for loop is used to iterate over a sequence (such as a list, tuple, or string) or other iterable objects. It allows you to execute a set of statements a specific number of times.

3. Syntax of for Loop: The general syntax of the for loop in Python is as follows:

```python
for element in sequence:
    # Code to be executed for each element
```


```
for element in sequence:
    # Code to be executed for each element
```


4. Example Program: Printing Numbers in a Range

```python
# Printing numbers from 1 to 5
for i in range(1, 6):
    print(i)
```


```
# Printing numbers from 1 to 5
for i in range(1, 6):
    print(i)
```


5. Example Program: Summing Numbers in a List

```python
# Summing numbers in a list
numbers = [10, 20, 30, 40, 50]
sum = 0

for num in numbers:
    sum += num

print("Sum:", sum)
```

```
# Summing numbers in a list
numbers = [10, 20, 30, 40, 50]
sum = 0

for num in numbers:
    sum += num

print("Sum:", sum)
```

6. while Loop: The while loop is used to repeatedly execute a block of code as long as a given condition is True.
7. Syntax of while Loop: The general syntax of the while loop in Python is as follows:

python Copy code

```
while condition:
    # Code to be executed while the condition is True
```

```
while condition:
    # Code to be executed while the condition is True
```

8. Example Program: Countdown Timer

python Copy code

```
# Countdown timer
count = 5

while count > 0:
    print(count)
    count -= 1

print("Blast off!")
```

```
# Countdown timer
count = 5
```

```
while count > 0:
    print(count)
    count -= 1

print("Blast off!")
```

9. Example Program: User Input Validation

python Copy code

```python
# User input validation
password = "secret"

while True:
    user_input = input("Enter the password: ")
    if user_input == password:
        print("Access granted!")
        break  # Exit the loop
    else:
        print("Access denied. Try again.")
```

```
# User input validation
password = "secret"

while True:
    user_input = input("Enter the password: ")
    if user_input == password:
        print("Access granted!")
        break  # Exit the loop
    else:
        print("Access denied. Try again.")
```

10. Infinite Loops and Loop Control: Be cautious with while loops, as they can potentially result in infinite loops if not properly controlled. You can use the break statement to exit a loop prematurely, and the continue statement to skip the current iteration and proceed to the next.

Arrays and Functions in Python

1. Introduction to Arrays: An array is a collection of elements, each identified by an index or a key. In Python, arrays are implemented using lists, which are a versatile and fundamental data structure. Lists can hold a mix of different data types and are mutable, meaning you can change their content after creation.

2. Creating and Accessing Lists: Lists are created by enclosing comma-separated values within square brackets [ ]. Elements in a list can be accessed using their index, starting from 0.

3. Example Program: Creating and Accessing a List

python Copy code

```python
# Creating a list
fruits = ["apple", "banana", "orange", "grape"]

# Accessing elements
print(fruits[0])  # Output: apple
print(fruits[2])  # Output: orange
```


```
# Creating a list
fruits = ["apple", "banana", "orange", "grape"]

# Accessing elements
print(fruits[0])  # Output: apple
print(fruits[2])  # Output: orange
```


4. Functions in Python: Functions are blocks of organized, reusable code that perform a specific task. They help improve code modularity, readability, and reusability. In Python, functions are defined using the def keyword.

5. Syntax of Function Definition: The general syntax of function definition in Python is as follows:

python Copy code

```python
def function_name(parameters):
    # Function code
    return result
```


```
def function_name(parameters):
  # Function code
```

```
    return result
```

6. Example Program: Simple Function

python Copy code

```python
# Function to add two numbers
def add_numbers(a, b):
    result = a + b
    return result

# Calling the function
sum = add_numbers(5, 3)
print("Sum:", sum)  # Output: Sum: 8
```

7. Passing Lists to Functions: You can pass lists as arguments to functions, allowing you to manipulate and process collections of data within the function.

8. Example Program: Function to Find Average

python Copy code

```python
# Function to calculate average
def calculate_average(numbers):
    total = sum(numbers)
    average = total / len(numbers)
    return average

# Calling the function
data = [12, 18, 24, 30, 36]
avg = calculate_average(data)
print("Average:", avg)  # Output: Average: 24.0
```

```
# Function to calculate average
def calculate_average(numbers):
    total = sum(numbers)
    average = total / len(numbers)
    return average
```

```
# Calling the function
data = [12, 18, 24, 30, 36]
avg = calculate_average(data)
print("Average:", avg)  # Output: Average: 24.0
```

9. Modifying Lists Inside Functions: Lists passed to functions can be modified within the function. Since lists are mutable, any changes made to the list inside the function will affect the original list outside the function as well.

10. Return Statements: Functions can return values using the return statement. You can return multiple values as a tuple.