Tushar Khitoliya

Python Programming Language

“The Easiest Python Programming Language Complete Crash Course”

2024 Syllabus Based Python Course

Basic of Programming, Everything about Python, Advantages of learnning Python.

Python & VS Code Installation In Windows & MacOS

Basic of Python, Syntax, Print Function, Comments, Terminal, Quotes.

Variables in Python, Basics of Datatypes.

Strings, Integers, Floats, Booleans, Lists, None, Dictionary, Tuples & Sets.

Loops in Python, Types of Loops, Loop control statements, Infinite Loop.

2024 Syllabus Based Python Course

Operators in Python,
Conditional Statements
(if, elif, else),
Try & Except Statement.

Functions in Python,
Function Invocation,
Return Statement,
Parameters,
Default Parameters.

PIP in Python,
Everything about pip,
Packages in Python,
Types of Packages.

File Operations
in Python,
Reading and
Writing Files,
Everything
about FileIO.

Class,
Object/Instance,
Attributes,
Methods,
Encapsulation,
Inheritance,
Polymorphism.

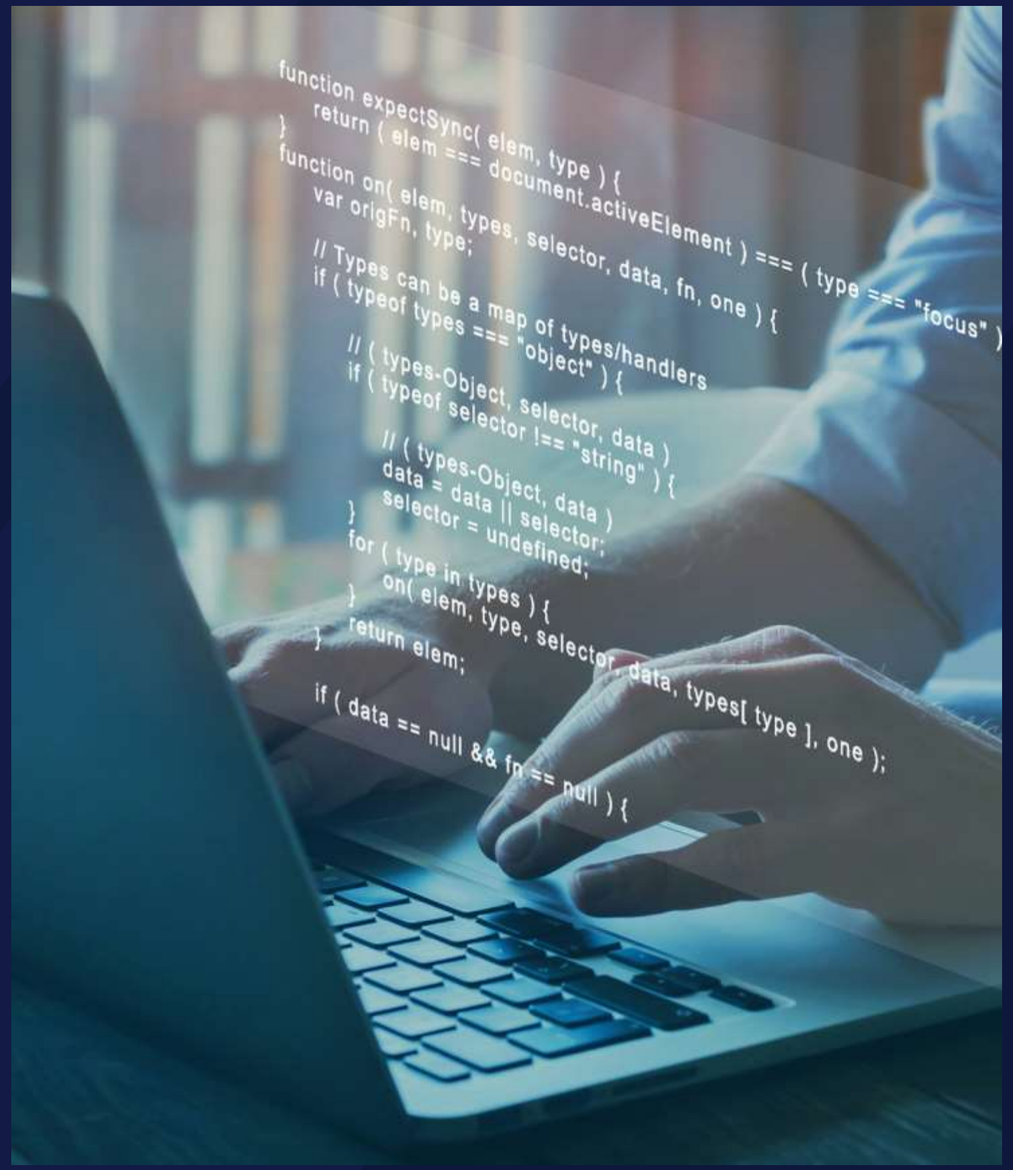
Practice,
Projects,
Kuch kaam ki
batein tumhare
liye.

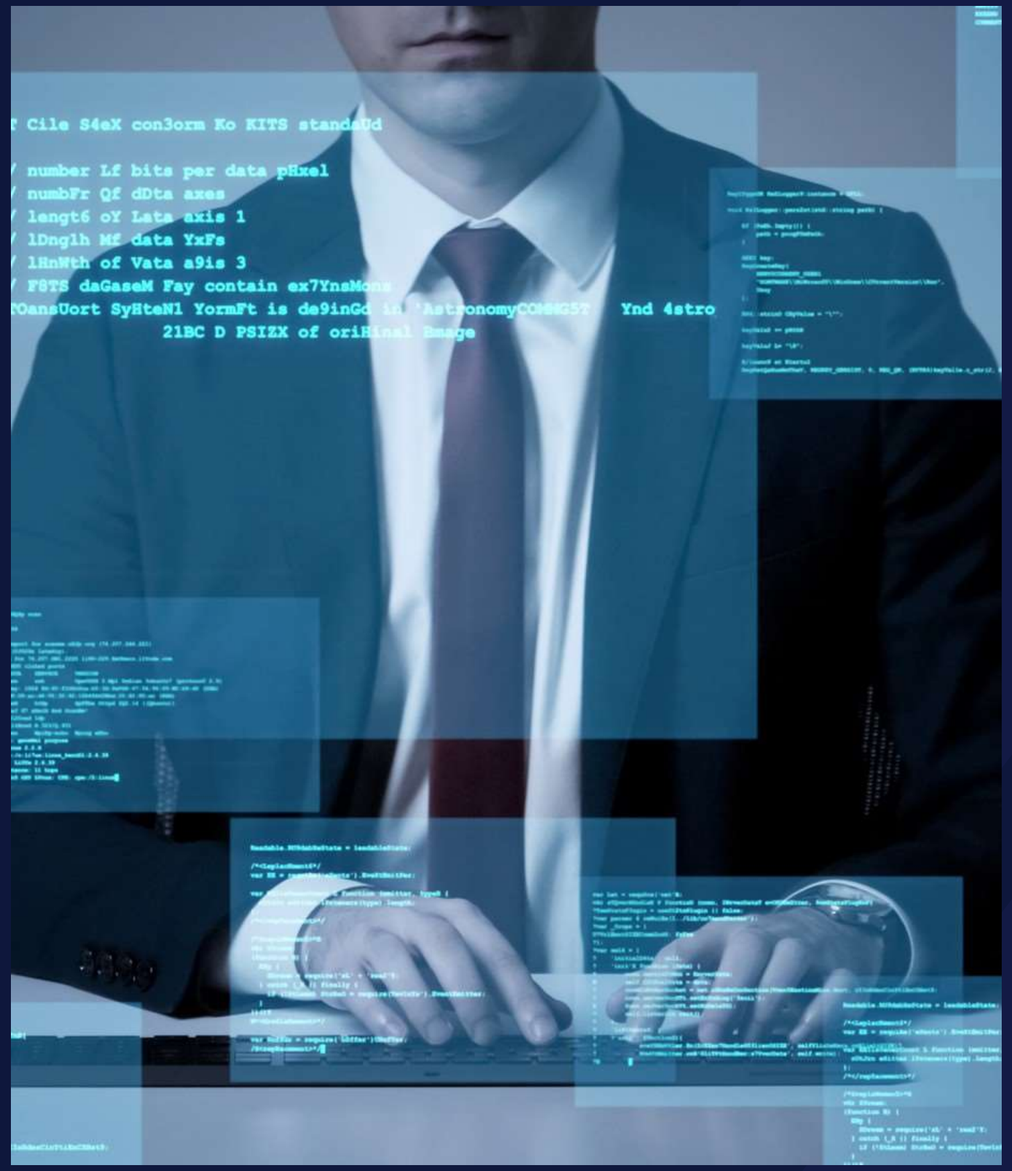
“This Course Is Designed To Cover Python Programming From The Very Basics”

“No Prerequisite, Just Start Learning”

What is Programming?

Programming refers to the process of creating sets of instructions (code) that a computer can understand and execute to perform specific tasks or solve problems.



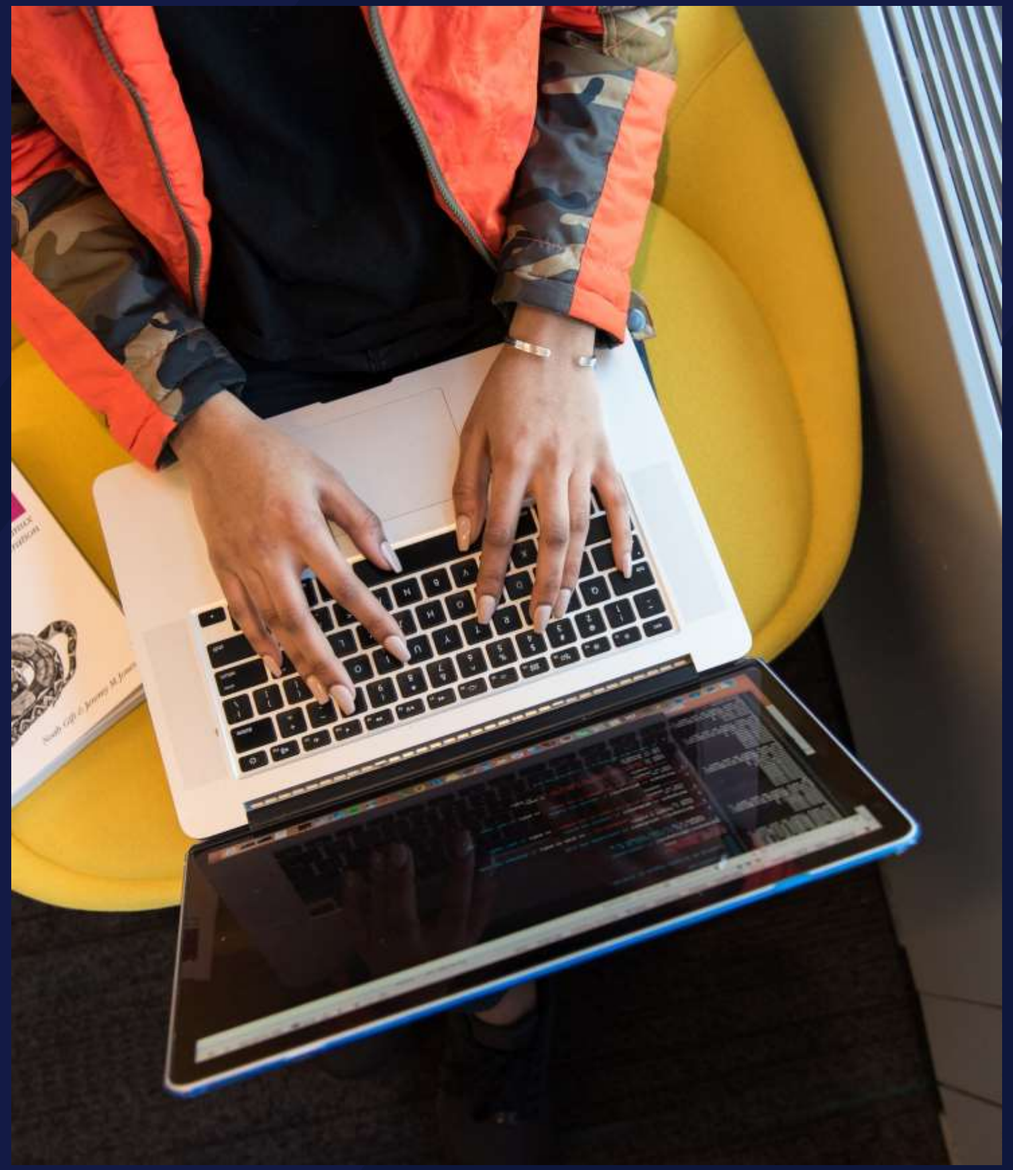


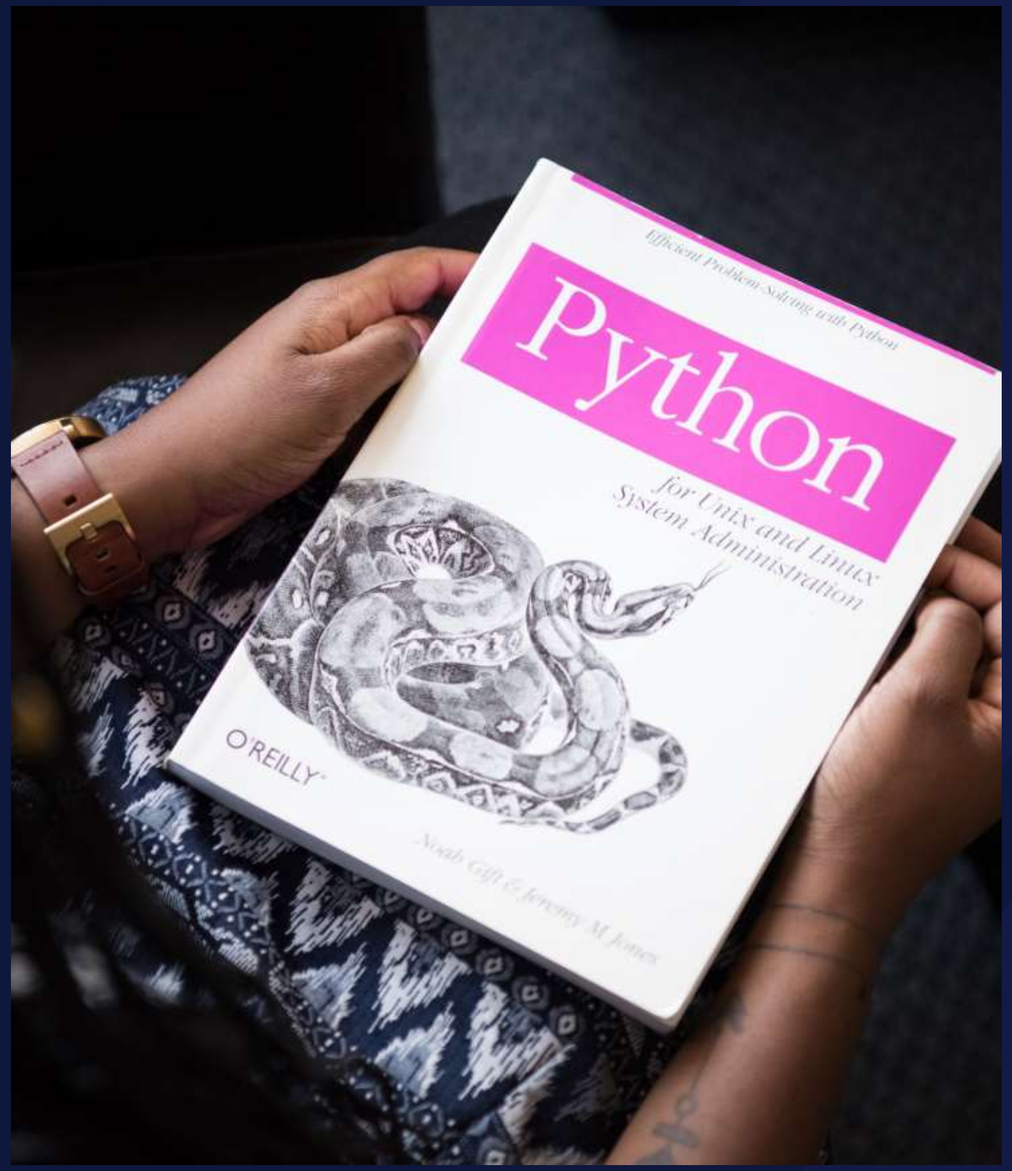
Who is Programmer?

A programmer, also known as a developer or software engineer, is an individual who writes, designs, creates, and maintains computer programs or software applications.

What is Python?

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





“History of Python”

Python, created by Guido van Rossum, was introduced in 1991. It aimed to be a readable, high-level programming language with a clear, minimalist syntax.

“Advantages of Learning Python”

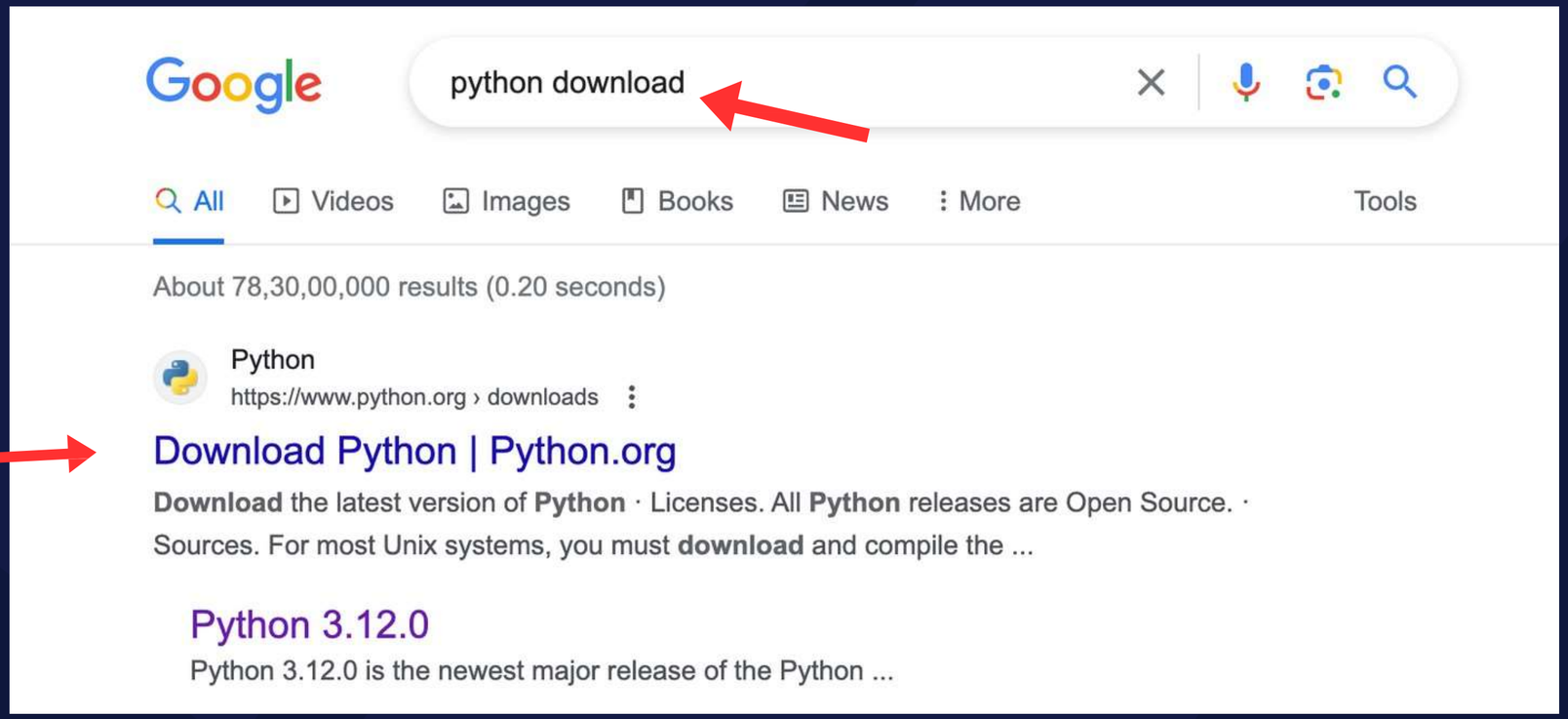
Python's simplicity, adaptability, and strong community support make it a favored choice across industries for its readability, versatility, and collaborative environment.

Tushar Khitoliya

“Python & VsCode Download”

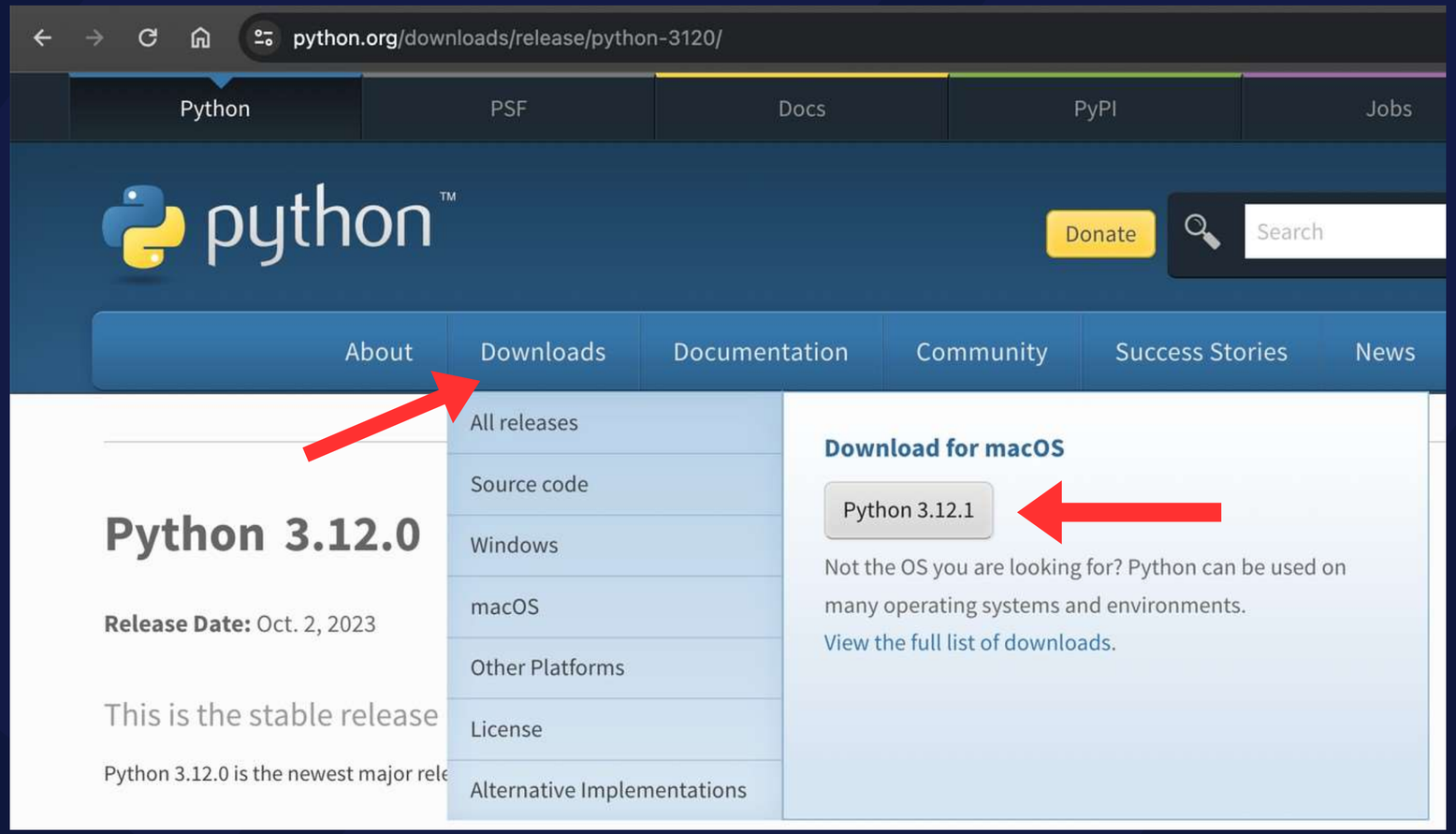
“Downloading Python”

1- Search “Python Download” On Google



2 - Click on the “Download Python”

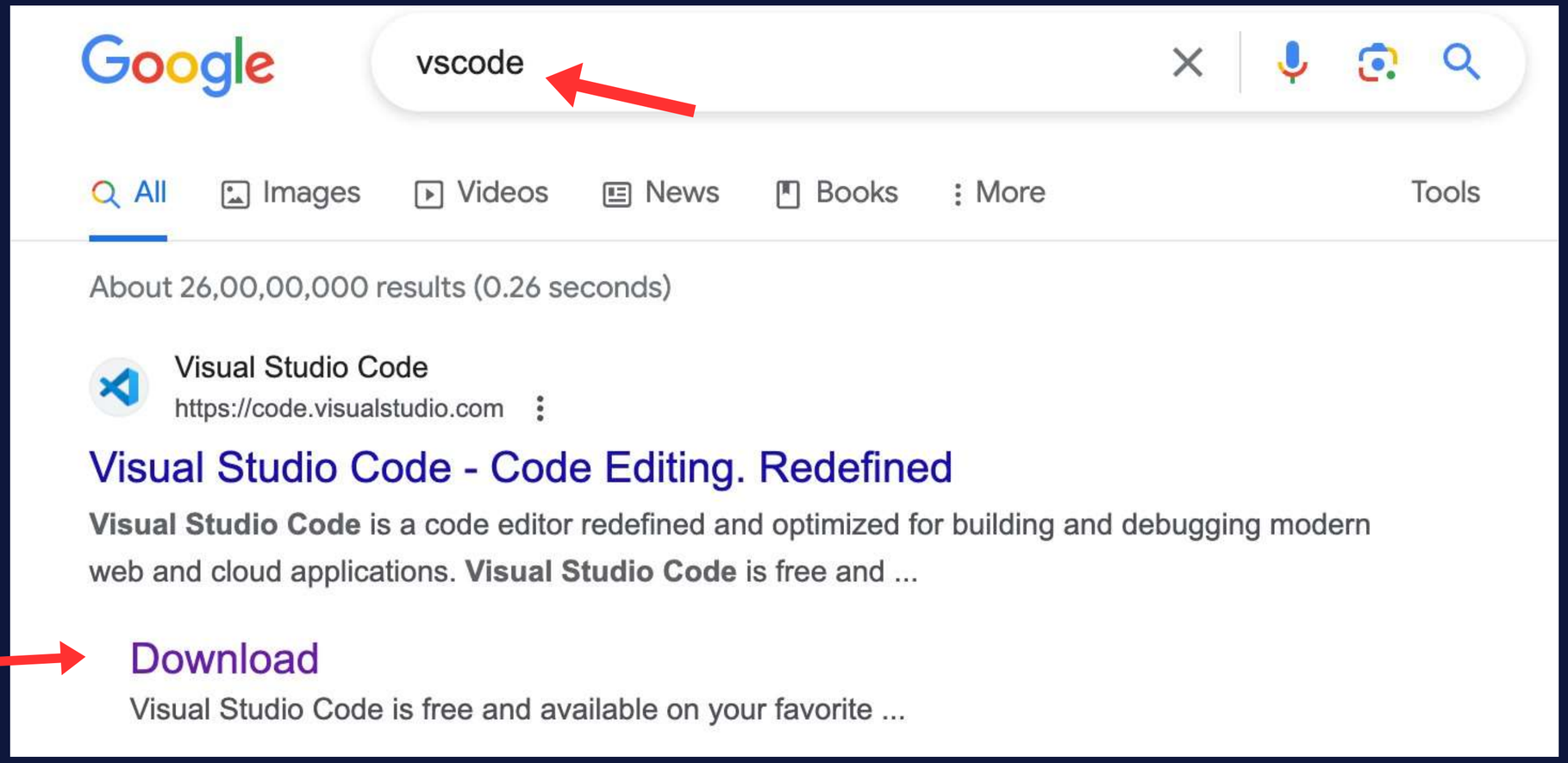
3 - Click on “Downloads” (Same Process for Windows Also)



4 - Click on the “Python 3.12.1” (Version may change in future)

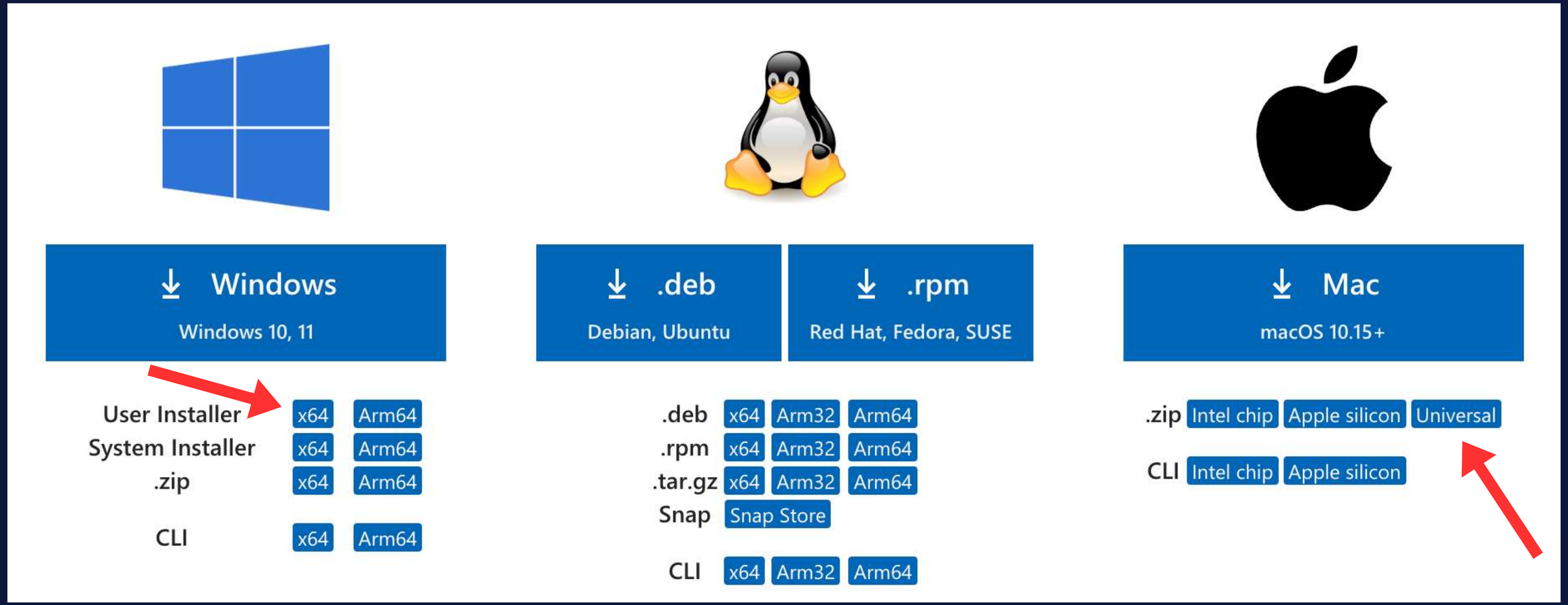
“Downloading VsCode”

1- Search “Vscode Download” On Google



2 - Click on the “Download”

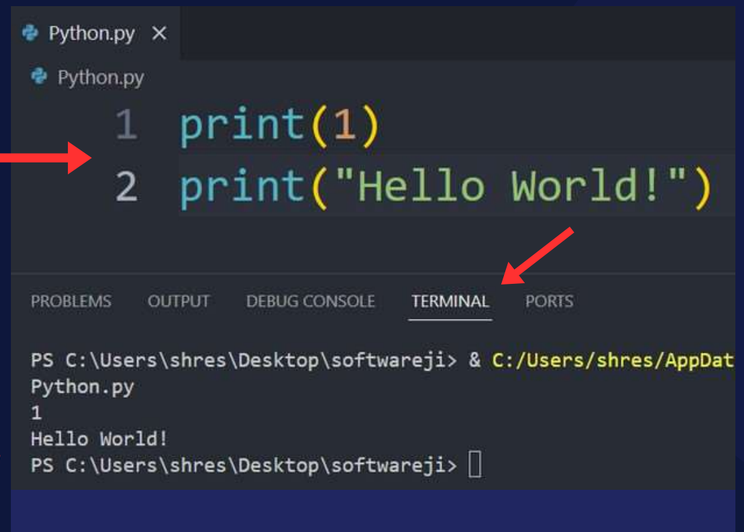
If you’re using Windows, Download Vscode for Windows.



If you’re using Macbook, Download Vscode for MacOs.

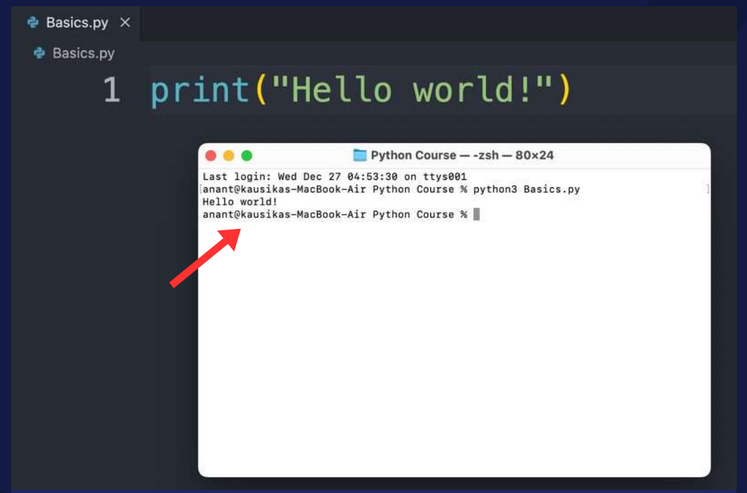
“Follow The Tutorial To Learn More About The Installation Process”

“Basics of Python”



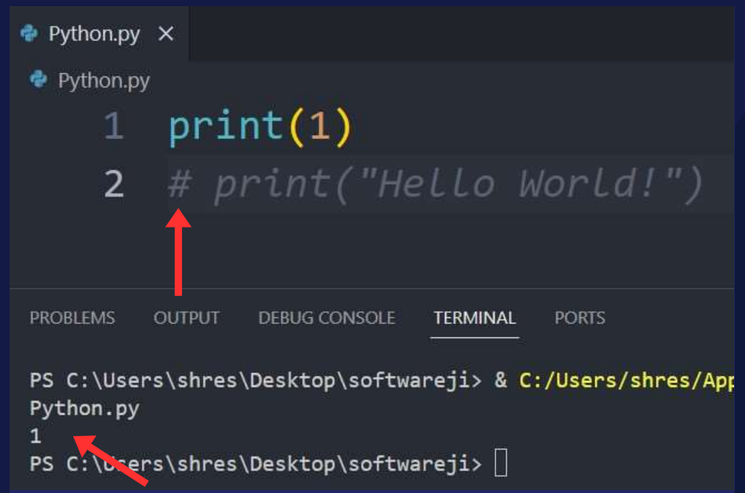
Print Function

In Python, the print() function is used to display output to the console or standard output. It outputs text or values of variables for the user or developer to see.



Terminal

The terminal, also known as the command prompt, shell, or console, provides an interface for interacting with a computer through text-based commands.



Comments

Comments in programming languages, including Python, are textual annotations within the code that are ignored by the interpreter or compiler.

“Basics of Python”

Syntax

Syntax in Python refers to the rules and structure that define how the Python programming language is written.

IDE

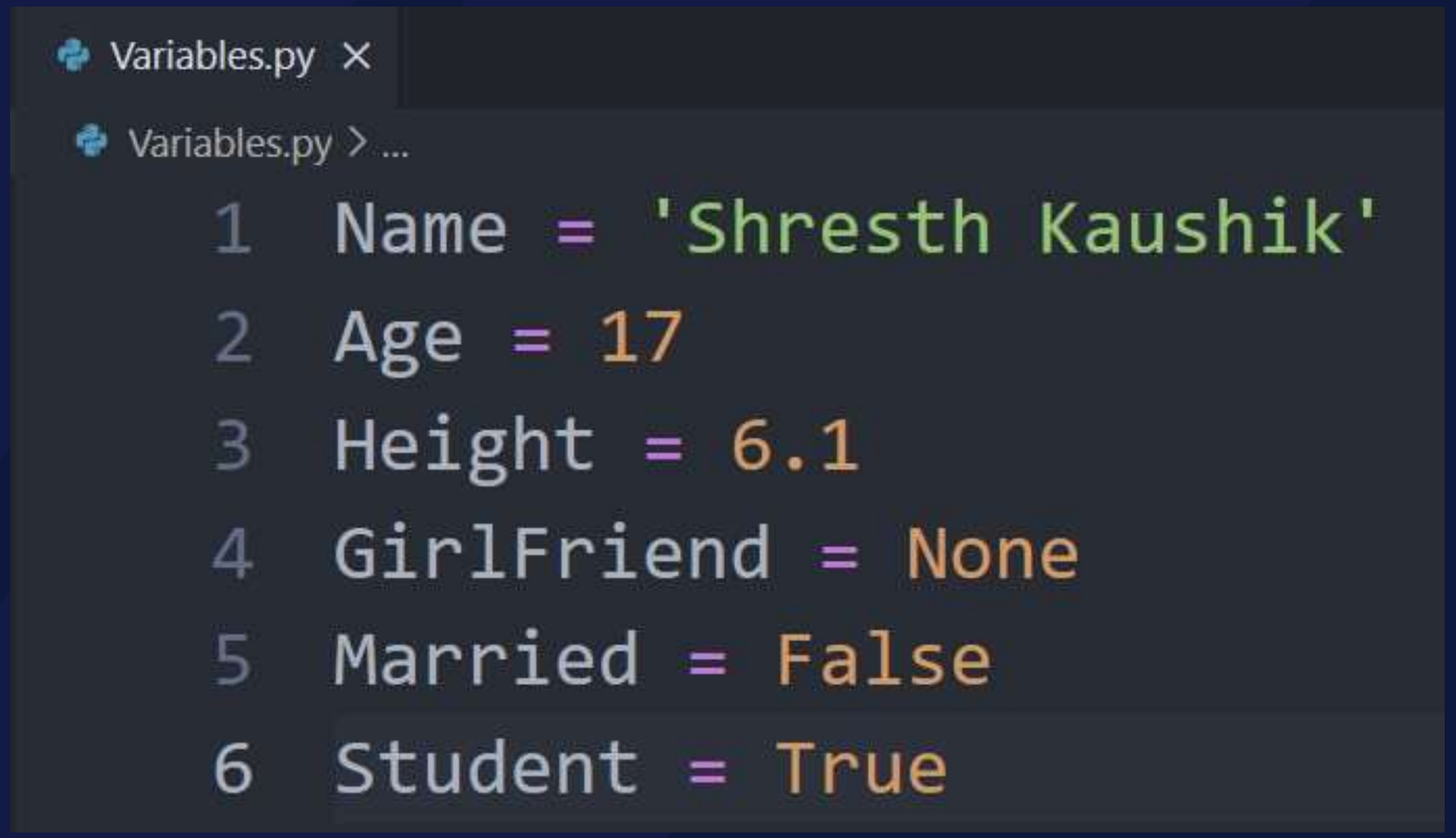
IDE stands for Integrated Development Environment. It's a software application that provides comprehensive facilities to computer programmers for development.

Quotes

In Python, quotes are used to denote strings. Strings are sequences of characters enclosed in either single (' ') or double (" ") quotes.

“Variables In Python”

In Python, variables are used to store data values. They act as containers for holding information that can be referenced and manipulated within a program. Python uses the equal sign (=) to assign values to variables. Must start with a letter (a-z, A-Z) or underscore (_).
Can include letters, numbers, and underscores. Python variables are case-sensitive



“Datatypes In Python”

In Python, data types represent the kind of value a variable can hold. Python is a dynamically typed language, which means you don't need to explicitly declare the data type of a variable. Instead, Python infers the data type based on the value assigned to it.

Datatypes - String, Float, Integer, List, Dictionary, Set, Tuple, Nonetype, Booleans.

“String In Python”

In Python, strings are a sequence of characters enclosed within single quotes (' '), double quotes (" "), or triple quotes (''' '''). Strings are immutable, meaning they cannot be changed after they are created.

You can create strings using single quotes, double quotes, or triple quotes for multiline strings.

Individual characters in a string can be accessed using indexing.

You can extract a specific portion (substring) of a string using slicing.

Strings can be concatenated using the + operator.

Python has numerous built-in string methods for various operations such as converting cases, finding substrings, replacing values, splitting strings, etc.

Strings.py > ...

```python
single_quoted_string = 'Hello, World!' # Single quotes
double_quoted_string = "Python Programming" # Double quotes
multiline_string = '''This is a
multiline
string.''' # Triple quotes for multiline strings

my_string = "Python"
print(my_string[0])  # Output: 'P'
print(my_string[-1])  # Output: 'n'

str1 = "Hello"
str2 = "World"
concatenated_str = str1 + ", " + str2 + "!"
print(concatenated_str)  # Output: 'Hello, World!'

my_string_1 = "Python Programming"
print(my_string_1[0:6])  # Output: 'Python'
print(my_string_1[7:])   # Output: 'Programming'
print(my_string_1[:6])   # Output: 'Python'
print(my_string_1[::-1])  # Output: 'gnimmargorP nohtyP' (reversed string)

my_string_2 = "Python Programming"
print(my_string_2.upper())    # Convert to uppercase
print(my_string_2.lower())    # Convert to lowercase
print(my_string_2.find('Pro'))  # Find the index of a substring
print(my_string_2.replace('Python', 'Java'))  # Replace part of the string
print(my_string_2.split())    # Split the string into a list based on whitespace

name = "Alice"
age = 30
formatted_string = f"My name is {name} and I am {age} years old."
print(formatted_string)  # Output: 'My name is Alice and I am 30 years old.'
```

“Integers In Python”

In Python, integers are whole numbers without any decimal point. They can be positive or negative. Python provides support for basic arithmetic operations and various functions for working with integers.

Integers are created without specifying the data type explicitly. They can be assigned directly to variables.

Python supports various arithmetic operations for integers such as addition, subtraction, multiplication, division, modulus, and exponentiation.

Python has two types of division operators. The single forward slash (/) performs floating-point division, while the double forward slash (//) performs integer division, resulting in a truncated integer value.

You can convert other data types to integers using int() function.

Integers.py

Integers.py > ...

```python
my_integer = 10
negative_integer = -5

a = 10
b = 5
# Addition
addition = a + b  # Result: 15
# Subtraction
subtraction = a - b  # Result: 5
# Multiplication
multiplication = a * b  # Result: 50
# Division
division = a / b  # Result: 2.0 (floating-point division)
# Modulus (Remainder)
modulus = a % b  # Result: 0 (remainder of division)
# Exponentiation
exponentiation = a ** b  # Result: 100000 (a raised to the power b)

result_float = 10 / 3  # Result: 3.33333...
result_integer = 10 // 3  # Result: 3 (truncated integer value)

num_str = "50"
converted_int = int(num_str)  # Result: 50
```

“Floats In Python”

In Python, floats represent decimal numbers or numbers with a fractional component. They are used to represent real numbers and are defined using a decimal point.
Floats can be created by directly assigning decimal numbers to variables.
Python supports various arithmetic operations for floats similar to integers, including addition, subtraction, multiplication, division, modulus, and exponentiation.
You can convert other data types to floats using the float() function.

Floats.py

Floats.py > ...

```python
my_float = 3.14
another_float = 6.75

a = 3.5
b = 2.0
# Addition
addition = a + b  # Result: 5.5
# Subtraction
subtraction = a - b  # Result: 1.5
# Multiplication
multiplication = a * b  # Result: 7.0
# Division
division = a / b  # Result: 1.75
# Modulus (Remainder)
modulus = a % b  # Result: 1.5
# Exponentiation
exponentiation = a ** b  # Result: 12.25

num_str = "3.5"
converted_float = float(num_str)  # Result: 3.5
```

“Booleans & Nonetype In Python”

In Python, both Booleans and None represent different types of values that hold specific meanings within the language.
Booleans represent the truth values **True** or **False**. They are used in logical operations, conditions, and control flow statements to make decisions based on whether an expression evaluates to **True** or **False**.
None is a special constant in Python that represents the absence of a value or a null value. It is used to indicate that a variable does not refer to any object or that a function does not return any value.

Booleans&None.py

```python
is_valid = True
is_greater = 10 > 5  # Result: True

is_valid = False
is_equal = 10 == 5  # Result: False

empty_variable = None

bool_value = bool(0)  # Result: False
bool_value = bool(10)  # Result: True

bool_value = bool(None)  # Result: False
bool_value = bool([])  # Result: False
bool_value = bool('Hello')  # Result: True

```

“Lists In Python”

In Python, a list is a versatile data structure that serves as an ordered collection of items. Lists are mutable, meaning their elements can be changed after creation. Lists are enclosed in square brackets [] and can contain elements of different data types, including integers, strings, floats, other lists, and more.Lists can be created by enclosing elements within square brackets and separating them with commas.

Elements in a list can be accessed using indexing. Indexing starts at 0 for the first element.

You can extract a specific portion (slice) of a list using slicing.

Lists are mutable, so you can change, add, or remove elements after creation.

Lists support various operations like concatenation (+), repetition (*), and length determination (len()).

List.py

List.py > ...

```python
# Creating Lists
my_list = [1, 2, 3, 4, 5]  # List of integers
mixed_list = ['apple', 3.14, True, 'banana']  # List of mixed data types
nested_list = [1, 'hello', [3, 4, 5], 'world']  # List containing another list

# Accessing Elements in a List
my_list_1 = ['apple', 'banana', 'cherry', 'date']
print(my_list_1[0])  # Output: 'apple'
print(my_list_1[2])  # Output: 'cherry'
print(my_list_1[-1])  # Output: 'date' (last element)

# List Slicing
my_list_2 = ['apple', 'banana', 'cherry', 'date']
print(my_list_2[1:3])  # Output: ['banana', 'cherry']
print(my_list_2[:2])   # Output: ['apple', 'banana']
print(my_list_2[2:])   # Output: ['cherry', 'date']

# Modifying Lists
my_list_3 = ['apple', 'banana', 'cherry']
my_list_3[1] = 'orange' # Changing Elements
print(my_list_3)  # Output: ['apple', 'orange', 'cherry']
my_list_4 = ['apple', 'banana', 'cherry']
my_list_4.append('date') # Adding Element
print(my_list_4)  # Output: ['apple', 'banana', 'cherry', 'date']
my_list_5 = ['apple', 'banana', 'cherry']
my_list_5.remove('banana') # Removing Elements
print(my_list_5)  # Output: ['apple', 'cherry']

# List Operations
list1 = [1, 2, 3]
list2 = [4, 5, 6]
concatenated_list = list1 + list2  # Concatenation
repeated_list = list1 * 3  # Repetition
length_of_list = len(list1)  # Length of the list
print(concatenated_list)
print(repeated_list)
print(length_of_list)
```

“Dictionary In Python”

In Python, a dictionary is a versatile and powerful data structure used to store collections of key-value pairs. Dictionaries are unordered, mutable, and enclosed in curly braces {}. Each key in a dictionary must be unique and immutable (such as strings, integers, or tuples), while values can be of any data type.

Dictionaries are created by defining key-value pairs within curly braces, separated by colons (:) and commas (,).

Values in a dictionary are accessed by providing the corresponding key in square brackets (**[]**).

Dictionaries are mutable, allowing you to change, add, or remove key-value pairs.

You can remove elements using the **del** keyword or the **pop()** method.

```python
# Creating a Dictionary
my_dict = {'name': 'Alice', 'age': 30, 'is_student': False}
# Dictionary with different data types as values
details = {
    'name': 'Bob',
    'age': 25,
    'height': 6.0,
    'grades': [85, 90, 75],
    'is_student': True }

# Accessing Values in a Dictionary
print(details['name'])  # Output: 'Bob'
print(details['age'])  # Output: 25
print(details['grades'])  # Output: [85, 90, 75]

# Modifying and Adding Elements in a Dictionary
details['age'] = 26
print(details['age'])  # Output: 26
details['address'] = '123 Street'
print(details)  # New key-value pair added

# Removing Elements from a Dictionary
del details['is_student'] # Del Statement
print(details)  # Output: 'is_student' key-value pair removed
address = details.pop('address') # Pop Function
print(address)  # Output: '123 Street' (value associated with 'address' key)
```

“Sets In Python”

In Python, a set is an unordered collection of unique elements. Sets are defined by enclosing comma-separated elements within curly braces {}. Sets are mutable but have no duplicate elements. They are particularly useful when dealing with unique elements or performing mathematical set operations.
Sets can be created using curly braces {} or by using the **set()** constructor.
Sets support various operations like adding elements, removing elements, and performing mathematical operations like union, intersection, difference, etc.

Sets.py

Sets.py > ...

```python
# Creating Sets
my_set = {1, 2, 3, 4, 5}  # Set with unique elements
another_set = {1, 2, 2, 3, 3, 4, 4}  # Duplicate elements are automatically removed
print(another_set)  # Output: {1, 2, 3, 4}
empty_set = set()  # Empty set

# Set Operations
my_set_1 = {1, 2, 3}
my_set_1.add(4)  # Adding a single element
print(my_set_1)  # Output: {1, 2, 3, 4}
my_set_1.update([5, 6, 7])  # Adding multiple elements
print(my_set_1)  # Output: {1, 2, 3, 4, 5, 6, 7}

my_set_2 = {1, 2, 3, 4, 5}
my_set_2.remove(3)  # Remove a specific element
print(my_set_2)  # Output: {1, 2, 4, 5}
my_set_2.discard(10)  # Discard an element that might not exist
print(my_set_2)  # Output: {1, 2, 4, 5}
popped_element = my_set_2.pop()  # Remove and return an arbitrary element
print(popped_element, my_set_2)  # Output: (any element), set without the popped element

set1 = {1, 2, 3, 4}
set2 = {3, 4, 5, 6}
# Union of sets
union_set = set1 | set2  # or set1.union(set2)
print(union_set)  # Output: {1, 2, 3, 4, 5, 6}
# Intersection of sets
intersection_set = set1 & set2  # or set1.intersection(set2)
print(intersection_set)  # Output: {3, 4}
# Difference between sets
difference_set = set1 - set2  # or set1.difference(set2)
print(difference_set)  # Output: {1, 2}
```

“Tuples In Python”

In Python, a tuple is an ordered and immutable collection of elements. Tuples are similar to lists, but once created, their elements cannot be changed, added, or removed. They are defined using parentheses () and can contain elements of different data types.

Tuples can be created by enclosing elements within parentheses and separating them with commas.

Elements in a tuple are accessed using indexing similar to lists. Indexing starts at 0 for the first element.

You can extract a specific portion (slice) of a tuple using slicing.

Tuples support operations like concatenation (+), repetition (*), and length determination (len()).

You can assign values from a tuple into multiple variables in a single line (tuple unpacking).

Tuples.py

```python
# Creating Tuples
my_tuple = (1, 2, 3, 4, 5)  # Tuple of integers
mixed_tuple = ('apple', 3.14, True)  # Tuple of mixed data types
nested_tuple = (1, 'hello', (3, 4, 5), 'world')  # Tuple containing another tuple

# Accessing Elements in a Tuple
my_tuple_1 = ('apple', 'banana', 'cherry', 'date')
print(my_tuple_1[0])  # Output: 'apple'
print(my_tuple_1[2])  # Output: 'cherry'
print(my_tuple_1[-1])  # Output: 'date' (last element)

# Tuple Slicing
my_tuple_2 = ('apple', 'banana', 'cherry', 'date')
print(my_tuple_2[1:3])  # Output: ('banana', 'cherry')
print(my_tuple_2[:2])   # Output: ('apple', 'banana')
print(my_tuple_2[2:])   # Output: ('cherry', 'date')

# Tuple Operations
tuple1 = (1, 2, 3)
tuple2 = (4, 5, 6)
concatenated_tuple = tuple1 + tuple2  # Concatenation
repeated_tuple = tuple1 * 3  # Repetition
length_of_tuple = len(tuple1)  # Length of the tuple
print(concatenated_tuple)
print(repeated_tuple)
print(length_of_tuple)

# Unpacking Tuples
my_tuple_3 = ('John', 'Doe', 30)
first_name, last_name, age = my_tuple_3
print(first_name, last_name, age)  # Output: 'John Doe 30'
```

“Loops In Python”

In Python, loops are used to execute a block of code repeatedly until a certain condition is met. There are mainly two types of loops: for loops and while loops.

Forloop.py

```python
# For Loop Syntax
sequence = [1,2,3,4,5,6]
for items in sequence:
    # Code block to be ex
    print(items)

#Iterating over a list
```

For Loop

for loops are typically used for iterating over sequences like lists, tuples, strings, and dictionaries, or any object that supports iteration.

whileloop.py

```python
# While Loop Syntax
count = 0
while count < 5:
    print(count)
    count += 1

# Using while loop with user input
user_input = ''
while user_input != 'quit':
    user_input = input("Enter 'quit' to exit: ")
    print("You entered:", user_input)

# Infinite Loop
while True:
    print("Subscribe!")
```

While Loop

while loops continue to execute a block of code as long as a specified condition is true.

whileloop.py

```python
# Example of Loop Control Statements
for i in range(5):
    if i == 3:
        break  # Exit the loop when i equals 3
    print(i)

for i in range(5):
    if i == 2:
        continue  # Skip the iteration when i equals 2
    print(i)
else:
    print("Loop completed without encountering a break statement.")
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\shres\Desktop\Python> & C:/Users/shres/AppData/Local/Pro
p.py
0
```

Loop Control Statements

Python provides control statements like break, continue, and else in loops.

“For Loop In Python”

The for loop in Python is used to iterate over a sequence (such as lists, tuples, strings, dictionaries, etc.) or an iterable object. It executes a block of code for each element in the sequence.
element is a variable that represents each item in the sequence during iteration.
sequence refers to the collection of elements being iterated.
The **for** loop is a fundamental construct in Python that allows you to iterate through collections or sequences, making it a powerful tool for handling repetitive tasks and data processing in various programming scenarios.

Forloop.py

Forloop.py > ...

```python
# For loop Syntax
sequence = [1,2,3,4,5,6]
for items in sequence:
    # Code block to be executed
    print(items)

#Iterating over a list
fruits = ['apple', 'banana', 'cherry']
for fruit in fruits:
    print(fruit)

# Iterating over a string
for char in "Python":
    print(char)

# Iterating over a range
for i in range(5):
    print(i)
```

“While Loop In Python”

The for loop in Python is used to iterate over a sequence (such as lists, tuples, strings, dictionaries, etc.) or an iterable object. It executes a block of code for each element in the sequence.
element is a variable that represents each item in the sequence during iteration.
sequence refers to the collection of elements being iterated.
The **for** loop is a fundamental construct in Python that allows you to iterate through collections or sequences, making it a powerful tool for handling repetitive tasks and data processing in various programming scenarios.

whileloop.py

whileloop.py > ...

```python
# While loop Syntax
count = 0
while count < 5:
    print(count)
    count += 1

# Using while loop with user input
user_input = ''
while user_input != 'quit':
    user_input = input("Enter 'quit' to exit: ")
    print("You entered:", user_input)

# Infinite Loop
while True:
    print("Subscribe!")
```

“Loop Control Statements In Python”

In Python, loop control statements are used to alter the flow of loops. There are three main loop control statements: break, continue, and else in loops.
The **break** statement is used to exit a loop prematurely based on a certain condition. When the **break** statement is encountered inside a loop, it immediately terminates the loop execution.
The **continue** statement is used to skip the current iteration of the loop and continue with the next iteration. It moves the control back to the loop's beginning.
Python allows an else block to be associated with a loop. The else block executes when the loop completes its normal iteration, i.e., without encountering a break statement.

whileloop.py

```python
# Example of Loop Control Statements
for i in range(5):
    if i == 3:
        break  # Exit the loop when i equals 3
    print(i)

for i in range(5):
    if i == 2:
        continue  # Skip the iteration when i equals 2
    print(i)
else:
    print("Loop completed without encountering a break statement.")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\shres\Desktop\Python> & C:/Users/shres/AppData/Local/Programs/
p.py
0
1
2
0
1
3
4
Loop completed without encountering a break statement.
```

“Operators In Python”

In Python, operators are special symbols or characters that perform operations on operands (variables, values, or expressions). Python supports various types of operators, including arithmetic, assignment, comparison, logical, identity, membership, and bitwise operators.

These operators perform mathematical operations.

- + (Addition)
- - (Subtraction)
- * (Multiplication)
- / (Division)
- % (Modulus - returns the remainder)
- ** (Exponentiation)
- // (Floor Division - returns the quotient)

Comparison Operators, Logical Operators Identity Operators, Membership Operators.

Operators.py > ...

```python
# Arithmetic operators
a = 10
b = 3
addition = a + b  # Addition
subtraction = a - b  # Subtraction
multiplication = a * b  # Multiplication
division = a / b  # Division
modulus = a % b  # Modulus (remainder)
exponentiation = a ** b  # Exponentiation
floor_division = a // b  # Floor Division (quotient)

# Assignment operators
x = 5
x += 3  # Equivalent to x = x + 3
print(f"Value of x after using compound assignment operator: {x}")

# Comparison operators
x = 10
y = 5
print(f"x == y: {x == y}")  # Equal to
print(f"x != y: {x != y}")  # Not equal to
print(f"x > y: {x > y}")  # Greater than
print(f"x < y: {x < y}")  # Less than
print(f"x >= y: {x >= y}")  # Greater than or equal to
print(f"x <= y: {x <= y}")  # Less than or equal to

# Logical operators
p = True
q = False
print(f"p and q: {p and q}")  # Logical AND
print(f"p or q: {p or q}")  # Logical OR
print(f"not p: {not p}")  # Logical NOT

# Identity operators
a = [1, 2, 3]
b = [1, 2, 3]
c = a
print(f"a is b: {a is b}")  # False (Different objects)
print(f"a is c: {a is c}")  # True (Same object)

# Membership operators
numbers = [1, 2, 3, 4, 5]
print(f"2 in numbers: {2 in numbers}")  # True (Value exists in list)
print(f"6 not in numbers: {6 not in numbers}")  # True (Value does not exist in list)
```

“Conditional Statement In Python”

Conditional statements in Python are used to perform different actions based on different conditions. The most common conditional statements in Python are if, elif (short for "else if"), and else.

ConditionalStatements.py

```python
#if Statement
x = 10
if x > 5:
    print("x is greater than 5")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\shres\Desktop\Python> & C:/Users/shres/AppData/Loc
nalStatements.py
x is greater than 5
PS C:\Users\shres\Desktop\Python>
```

IF Statement

The if statement is used to execute a block of code if a condition is true.

ConditionalStatements.py

```python
grade = 75

if grade >= 90:
    print("Grade is A")
elif grade >= 80:
    print("Grade is B")
elif grade >= 70:
    print("Grade is C")
else:
    print("Grade is below C")
```

Elif Statement

The elif statement allows you to check additional conditions if the preceding if statement's condition is false. It stands for "else if."

ConditionalStatements.py

```python
grade = 75

if grade >= 90:
    print("Grade is A")
elif grade >= 80:
    print("Grade is B")
elif grade >= 70:
    print("Grade is C")
else:
    print("Grade is below C")
```

Else Statement

The else statement is used to execute a block of code when the preceding if or elif conditions are false.

“Try & Except In Python”

The try and except blocks in Python are used for exception handling, allowing you to manage and handle errors or exceptions that may occur during the execution of code. They help prevent the program from crashing due to unexpected errors.
The **try** block contains the code where exceptions may occur, and the **except** block catches and handles those exceptions.
You can handle multiple types of exceptions by adding multiple **except** blocks.
Using try and except blocks allows you to gracefully handle exceptions and continue program execution, providing a way to deal with unexpected errors without causing the program to crash.

TryandExcept.py

TryandExcept.py > ...

```python
# Example
try:
    num = int(input("Enter a number: "))
    result = 10 / num
    print("Result:", result)
except ZeroDivisionError:
    print("Error: Division by zero")
except ValueError:
    print("Error: Invalid input. Please enter a valid number")
```

“Functions In Python”

In Python, a function is a block of reusable code that performs a specific task or set of tasks. Functions provide modularity, allowing you to break down your code into smaller, manageable parts, making it easier to read, understand, and maintain.

You can define a function in Python using the **def** keyword followed by the function name and parentheses containing any parameters. The function body is indented below.

Functions can take parameters (inputs) to perform operations based on the provided values.

Functions can return a value using the return statement. This value can be stored in a variable or used in other parts of the code.

You can specify default values for parameters, allowing the function to be called without providing those arguments. Also, arguments can be passed using keyword syntax.

Functions.py

```python
# Example of a Simple Function
def greet():
    print("Hello, welcome!")
# Calling the function
greet()  # Output: "Hello, welcome!"

# Function Parameters
def greet_user(name):
    print(f"Hello, {name}!")
# Calling the function with an argument
greet_user("Alice")  # Output: "Hello, Alice!"

# Return Statement
def add_numbers(a, b):
    return a + b
# Calling the function and storing the returned value
result = add_numbers(3, 5)
print("Result:", result)  # Output: 8

# Default Parameters and Keyword Arguments
def greet_person(name="Guest"):
    print(f"Hello, {name}!")
# Calling the function without providing an argument
greet_person()  # Output: "Hello, Guest!"
# Calling the function with a keyword argument
greet_person(name="Alice")  # Output: "Hello, Alice!"
```

“PIP In Python”

In Python, pip is a package manager used for installing and managing additional libraries or packages that extend the functionality of Python. It stands for "Pip Installs Packages." pip simplifies the process of downloading and installing external packages from the Python Package Index (PyPI) or other package indexes.

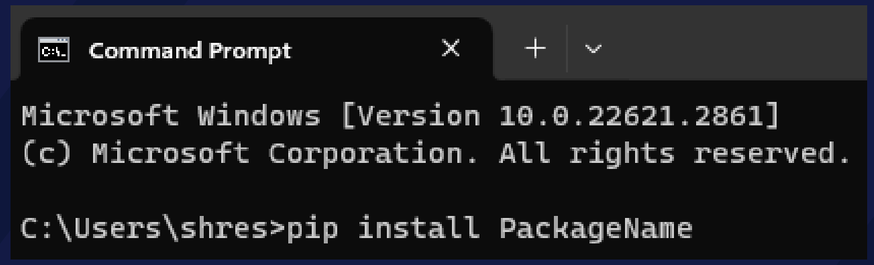
To install a package, you can use the **pip install** command followed by the name of the package.

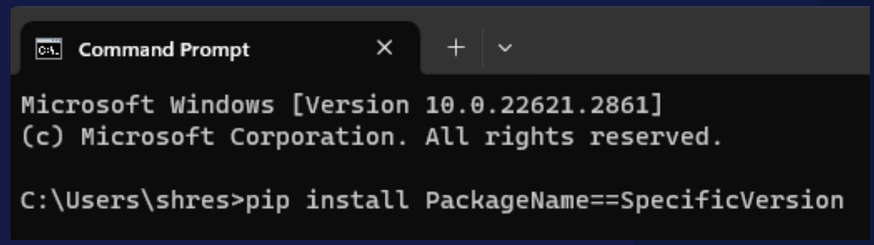
You can install a specific version of a package by specifying the version number along with the package name.

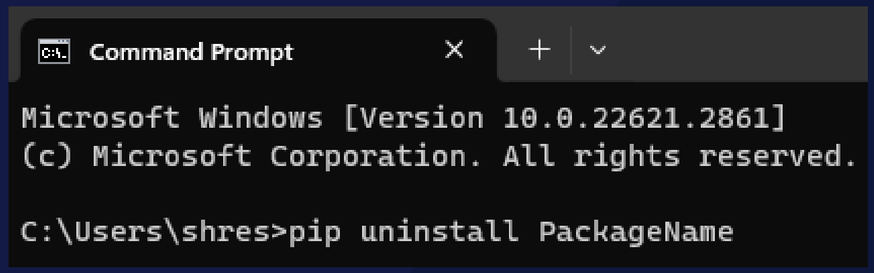
To uninstall a package, use the **pip uninstall** command followed by the name of the package.

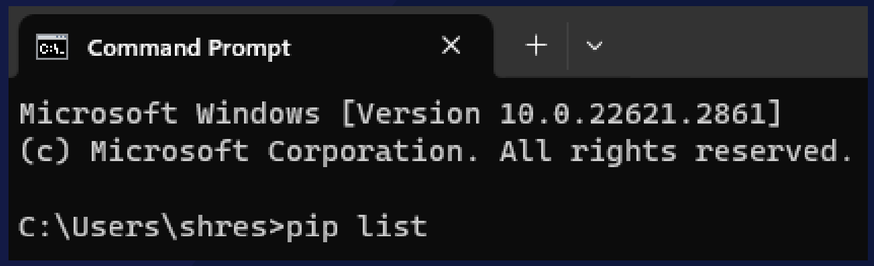
To view a list of installed packages and their versions, use the pip list command.

pip install requests # Installs the 'requests' package pip install numpy==1.21.3 # Installs a specific version of 'numpy' pip uninstall requests # Uninstalls the 'requests' package pip list # Lists all installed packages









“Packages In Python”

Python packages are collections of modules (Python files) containing reusable code that can be imported and used in Python programs. There are two types of packages

- Inbuilt Packages
- External Packages

These packages are available for various purposes, such as data manipulation, web development, machine learning, etc.

Popular Python packages include:

- **numpy** (for numerical computations)
- **pandas** (for data manipulation and analysis)
- **requests** (for making HTTP requests)
- **matplotlib** and **seaborn** (for data visualization)
- **scikit-learn** (for machine learning)
- **flask** and **Django** (for web development)

Once a package is installed, you can import it into your Python script or interactive session using the **import** statement.

“File Operations In Python”

File operations in Python involve reading from and writing to files. Python provides built-in functions and methods to interact with files on the file system.

The **open()** function is used to open a file. It takes two parameters: the file name/path and the mode in which the file should be opened ("**r**" for read, "**w**" for write, "**a**" for append, "**r+**" for read and write, etc.).

Once a file is opened in read mode ("**r**"), you can use various methods to read its contents.

When a file is opened in write mode ("**w**"), you can write data to it using the **write()** method.

When a file is opened in append mode ("**a**"), you can add content to the end of the file using the **write()** method.

It's good practice to close a file after performing operations on it using the **close()** method. However, using a context manager (**with** statement) automatically closes the file when the block inside the **with** statement is exited.

FileOperations.py

FileOperations.py > ...

```python
# Syntax
file = open("filename.txt", "mode")

# Reading from a File
with open("filename.txt", "r") as file:
    content = file.read()
    print(content)

# Writing to a File
with open("output.txt", "w") as file:
    file.write("This is a sample text.")

# Appending to a File
with open("filename.txt", "a") as file:
    file.write("\nThis is appended text.")

# Closing a File
file = open("filename.txt", "r")
file.close()  # Close the file when done

```

“Object-oriented programming In Python”

Object-oriented programming (OOP) is a programming paradigm that uses objects and classes to design and structure code. In Python, OOP allows you to create reusable and modular code by defining classes and objects.

Class: A class is a blueprint that defines the attributes (properties) and methods (functions) common to all objects of that class.

Object: An object is an instance of a class. It's a realization of the class, which contains its own unique data and methods.

Encapsulation refers to the bundling of data (attributes) and methods that operate on the data within a single unit (class). It helps in data hiding and abstraction.

nheritance allows a class (called a child or subclass) to inherit properties and behaviors (attributes and methods) from another class (called a parent or superclass).

Polymorphism allows objects of different classes to be treated as objects of a common parent class. It enables methods to be overridden in a subclass.

OOPS.py

OOPS.py > ...

```python
class Dog: # Define a class
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def bark(self):
        return "Woof!"

my_dog = Dog("Buddy", 3) # Create an object (instance of the Dog class)
print(my_dog.name)  # Output: "Buddy"
print(my_dog.age)   # Output: 3
print(my_dog.bark())  # Output: "Woof!"

# Example of Inheritance
class Animal: # Parent class
    def sound(self):
        pass
class Dog(Animal): # Child class inheriting from Animal
    def sound(self):
        return "Woof!"
class Cat(Animal): # Child class inheriting from Animal
    def sound(self):
        return "Meow!"

# Example of Polymorphism
def make_sound(animal): # Polymorphic behavior
    return animal.sound()
dog = Dog()
cat = Cat()
print(make_sound(dog))  # Output: "Woof!"
print(make_sound(cat))  # Output: "Meow!"
```

Tushar Khitoliya

“Congratulations Guys, You’ve Completed Python Syllabus”