Lab#1

ESHA ALI

9339

Introduction to Tools Setup and Python Programming

Lab Objectives:

By the end of this lab, students will be able to:

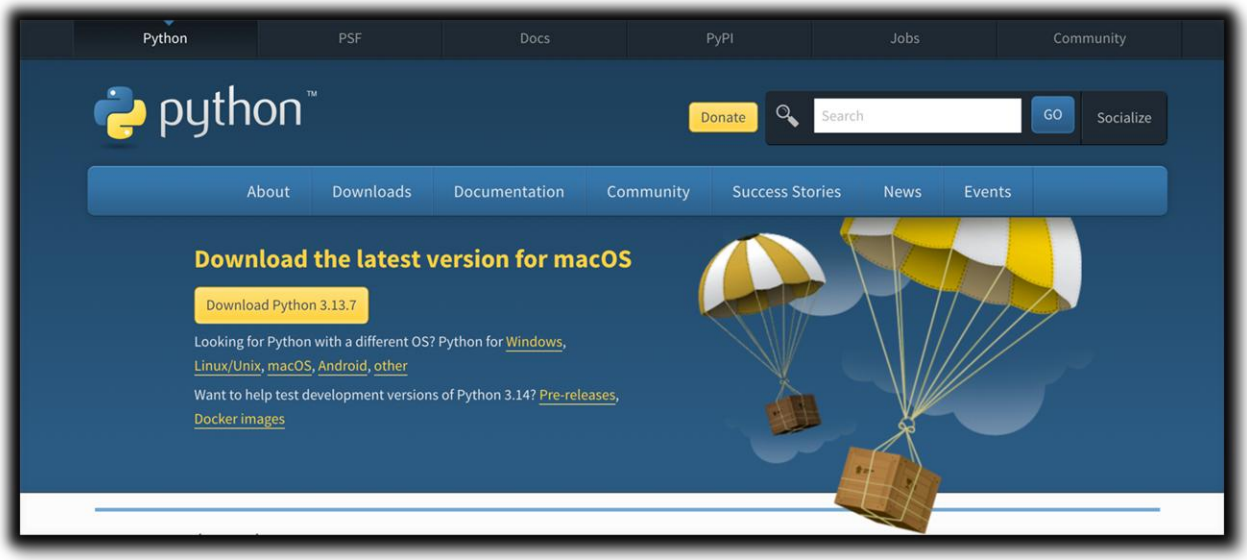
- Set up a complete python development environment and write functional python programs using variables, control structures, and functions
- Apply debugging techniques and problem-solving skills to create a complete application that integrates core programming concepts

Part 1: Setting Up Development Tools

1.1 Install Python

Step 1:

1. Go to python.org Download Python
2. Click "Download Python 3.x" (latest version)



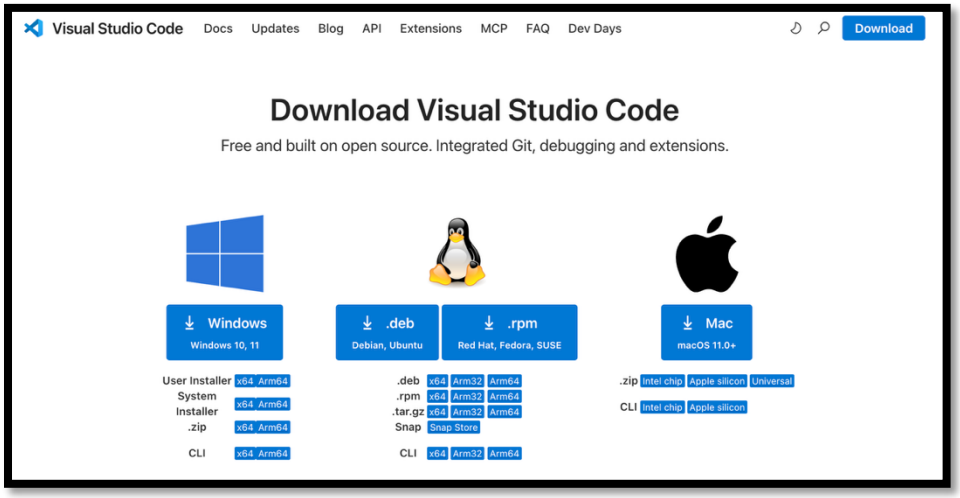
3. Run the installer
4. **Important:** Check "Add Python to PATH" during installation.
5. Installation Complete.

Step 2: Verify Installation

1. Open Command Prompt (Windows) or Terminal (Mac/Linux)
2. Type: python --version
3. You should see the Python version number

1.2 Install a Code Editor/IDE

Choose one of the following:

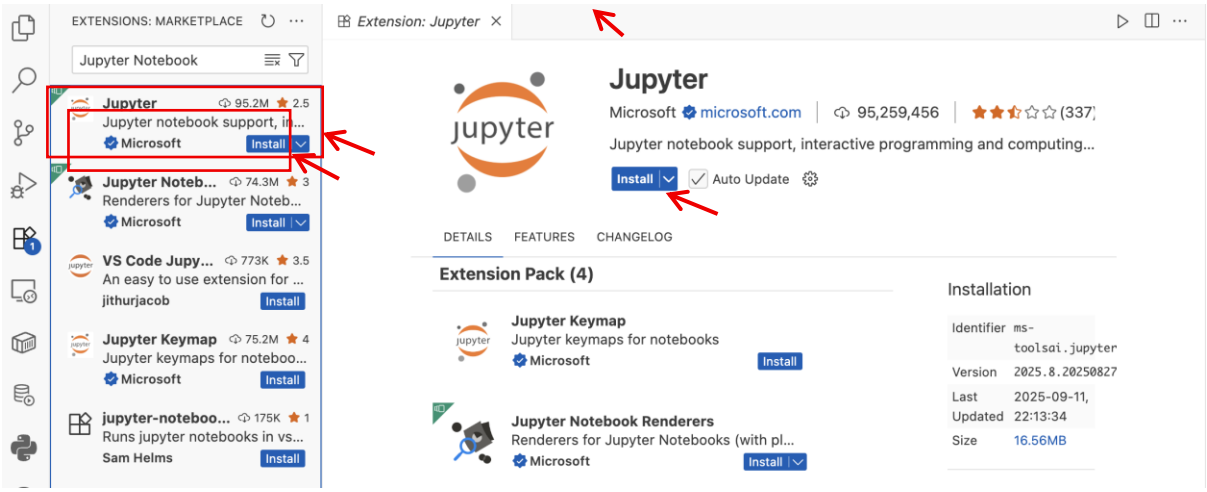


1. Visual Studio Code (Recommended)
2. Download from code.visualstudio.com.
3. Install the application for Windows / Linux / Mac.

4. Install Python extension, Open VS Code
5. Go to Extensions or press Ctrl+Shift+X

6. Search "Python" and install the Microsoft Python extension

7. Install Jupyter Notebook extension in VS code



Part 2: Python Programming

2.1 Your First Python Program

Step 1:

1. Open your code editor
2. Create a new file called `Lab#1.ipynb`
3. Type the following code:

```
# My first Python program
print("Hello, World!")
print("Welcome to Python programming!")
```

Step 2: Run the program

- Save the file
- Open terminal/command prompt
- Navigate to the file location: cd path/to/your/file
- Run: python hello.py

Expected Output:

```
Hello, World!
Welcome to Python programming!
```

2.2 Variables and Data Types

Variables:

Variables are named storage areas in the memory of a computer program, to store data values that may be altered or manipulated in the course of executing a computer program. They enable programmers to save, edit and access information with ease without directly addressing the raw memory addresses.

Data Types:

Types of data determine the type of data a variable may contain, i.e. integer(whole numbers), float/doubles(descimal numbers), strings(text), booleans(true/false). They make the computer assign appropriate memory space and carry out computer operations on the stored data properly.

Practice with different data types:

Example Code:

```
# Variables and Data Types

# Strings

name = "Alice"
message = 'Python is fun!'
multiline = """This is a
multiline string"""

print("Name:", name)
print("Message:", message)
print("Multiline:", multiline)

# Numbers
age = 25
height = 5.8
temperature = -10.5

print("Age:", age)
print("Height:", height)
print("Temperature:", temperature)

# Booleans
is_student = True
has_license = False

print("Is student:", is_student)
print("Has license:", has_license)
```

2.3 Basic Operations

Basic operations in programming are the fundamental actions performed on data to process and manipulate it. They are mainly divided into two categories:

- **Arithmetic operations**, which deal with numbers and include addition, subtraction, multiplication, division, floor division, modulus, and exponentiation.
- **String operations**, which deal with text and include concatenation, repetition, indexing, slicing, and finding length. These operations form the foundation of programming, allowing developers to work with both numerical and textual data efficiently.

Example Code:

```
# Basic Operations

# Arithmetic Operations
a = 10
b = 3

print("Addition:", a + b)
print("Subtraction:", a - b)
print("Multiplication:", a * b)
print("Division:", a / b)
print("Floor Division:", a // b)
print("Modulus:", a % b)
print("Exponentiation:", a ** b)

# String Operations
first_name = "John"
last_name = "Doe"
full_name = first_name + " " + last_name
print("Full name:", full_name)
print("Name repeated:", first_name * 3)
```

2.4 User Input

The process of accepting data or information that is directly entered by the user when executing a program is **known as user input**. It opens the line between the program and the user and the user passes information (such as numbers, text, or selections) that the program interprets to generate an output. To give an example, in Python, user input is accepted using **input() function**.

Example Code:

```
# Getting User Input

# Basic input
name = input("Enter your name: ")
print("Hello,", name)

# Converting input to numbers
age_str = input("Enter your age: ")
age = int(age_str)
print("You are", age, "years old")

# Better way with error handling
try:
    height = float(input("Enter your height in
meters: "))
    print("Your height is", height, "meters")
except ValueError:
    print("Invalid input! Please enter a number.")

# Multiple inputs
print("Enter three numbers:")
num1 = float(input("First number: "))
num2 = float(input("Second number: "))
num3 = float(input("Third number: "))

average = (num1 + num2 + num3) / 3
print("Average:", average)
```

Part 3: Control Structures

Basic blocks of programming include control structures which handle the execution flow of a program. They allow decision making, repetition and branching to make programs dynamic and effective. The primary conditional statements are if, if-else, if-elif-else to make decisions based on a condition.

Example Code:

```
# Conditional Statements

# Simple if statement
temperature = float(input("Enter temperature in Celsius: "))

if temperature > 30:
    print("It's hot outside!")
elif temperature > 20:
    print("It's warm outside!")
elif temperature > 10:
    print("It's cool outside!")
else:
    print("It's cold outside!")

# Multiple conditions
age = int(input("Enter your age: "))
has_license = input("Do you have a driver's license? (yes/no):
").lower() == "yes"

if age >= 18 and has_license:
    print("You can drive!")
elif age >= 18:
    print("You can get a license!")
else:
    print("You're too young to drive.")

# Grade calculator
score = float(input("Enter your test score (0-100): "))

if score >= 90:
    grade = "A"
elif score >= 80:
    grade = "B"
elif score >= 70:
    grade = "C"
elif score >= 60:
    grade = "D"
else:
    grade = "F"

print(f"Your grade is: {grade}")
```

3.2 Loops

Loops enable a group of statements to be repeatedly re-executed until a specified condition is fulfilled. They facilitate the minimization of redundancy, through automation of repetitive operations and make the code more concise and efficient.

- **For Loops:** The primary ones are **for loops**, which go through a series or a range of values.
- **While Loops:** While loops which are repeated till a condition is fulfilled
- **Loop Control:** Loop control statements such as break, continue are employed to control loop flow.

Example Code:

```
# Loops in Python

# For loop with range
print("Counting from 1 to 5:")
for i in range(1, 6):
    print(i)

# For loop with list
fruits = ["apple", "banana", "orange", "grape"]
print("\nFruits:")
for fruit in fruits:
    print(f"I like {fruit}")

# While loop
print("\nCountdown:")
count = 5
while count > 0:
    print(count)
    count -= 1
print("Blast off!")

# Nested loops - multiplication table
print("\nMultiplication table for 5:")
for i in range(1, 11):
    result = 5 * i
    print(f"5 x {i} = {result}")
```

Part 4: Functions and Debugging

4.1 Functions

Functions refer to similar blocks of program code that can be reused to do a particular task in a program. They assist in the organisation of code, lower the level of repetition, enhance the readability, and maintainability. A function has input (also known as parameters), carries out some processing and may have an output (also known as return value). Most programming languages require functions to be written once and to be invoked many times any time they are required.

Example Code:

```
# Functions in Python

# Simple function
def greet():
    print("Hello from a function!")

# Function with parameters
def greet_person(name):
    print(f"Hello, {name}!")

# Function with return value
def add_numbers(a, b):
    return a + b

# Function with default parameters
def introduce(name, age=18, city="Unknown"):
    return f"My name is {name}, I'm {age} years
old, from {city}"

# Function with multiple return values
def get_name_parts(full_name):
    parts = full_name.split()
    if len(parts) >= 2:
        return parts[0], parts[-1]
    else:
        return parts[0], ""

# Functions Calling
greet()
greet_person("Alice")

result = add_numbers(5, 3)
print(f"5 + 3 = {result}")

intro1 = introduce("Bob")
intro2 = introduce("Carol", 25, "New York")
print(intro1)
print(intro2)

first, last = get_name_parts("John Doe Smith")
```

4.2 Basic Debugging

Basic debugging process refers to the process of discovering, analyzing and correcting bugs or errors within a program in order to make the program execute properly. It includes reading error messages, tracing values with print statements, checking the logic of code and testing one part at a time. The debugging assists the programmer in locating syntax, logical or runtime errors and making sure that the program delivers the expected results.

Example Code:

```
# Simple Calculator Program


# Functions for basic arithmetic operations
# Addition Fuction
def add(a, b):
    return a + b
# Subtraction Function
def subtract(a, b):
    return a - b
# Multiply Funtion
def multiply(a, b):
    return a * b
# Divide Function
def divide(a, b):
    if b != 0:   # Avoid dividing by zero
        return a / b
    else:
        return "Error: Division by zero!"


# Program starts here
print("Basic Calculator")
```

Lab Tasks

1. Write a program that asks the user for details like name, age, email, etc., and then displays them.

```
name = input("Enter your name: ")

age = input("Enter your age: ")

email = input("Enter your email: ")



print("\n--- Your Details ---")

print("Name:", name)

print("Age:", age)

print("Email:", email)
```

```
===== RESTART: C:/Users/hp/AppData/Local
Enter your name: esha
Enter your age: 21
Enter your email: eshaa7848@gmail.com

--- Your Details ---
Name: esha
Age: 21
Email: eshaa7848@gmail.com
>> S
```

2. Write a program that calculates the area of a rectangle using:

   Formula: **Area = Length X Width**

```
length = float(input("Enter the length of the rectangle: "))

width = float(input("Enter the width of the rectangle: "))



area = length * width



print("The area of the rectangle is:", area)
```

```
===== RESTART: C:/Users/hp/AppData/Local/
Enter the length of the rectangle: 4
Enter the width of the rectangle: 3
The area of the rectangle is: 12.0
```

3. Write a program to build a simple To-Do List using functions to:

- Add tasks
- View tasks
- Remove tasks

```
tasks = []



def add_task():

    task = input("Enter a task: ")

    tasks.append(task)

    print("Task added!")



def view_tasks():
```

```
    print("\n--- To-Do List ---")
    if not tasks:
        print("No tasks added yet.")
    else:
        for i, task in enumerate(tasks, 1):
            print(f"{i}. {task}")

def remove_task():
    view_tasks()
    if tasks:
        num = int(input("Enter the task number to remove: "))
        if 1 <= num <= len(tasks):
            tasks.pop(num - 1)
            print("Task removed!")
        else:
            print("Invalid task number!")

while True:
    print("\n--- To-Do List Menu ---")
    print("1. Add Task")
    print("2. View Tasks")
```

```
print("3. Remove Task")

print("4. Exit")



choice = input("Enter your choice: ")



if choice == "1":

    add_task()

elif choice == "2":

    view_tasks()

elif choice == "3":

    remove_task()

elif choice == "4":

    print("Exiting program...")

    break

else:

    print("Invalid choice! Try again.")
```

```
--- To-Do List Menu ---
1. Add Task
2. View Tasks
3. Remove Task
4. Exit
Enter your choice:
```

4. Create a simple number guessing game that incorporates everything.

```
import random

name = input("Enter your name: ")
print(f"Welcome {name}! Let's play a number guessing game.")

number = random.randint(1, 100)
tries = 0

while True:
    guess = int(input("Enter your guess (1-100): "))
    tries += 1

    if guess > number:
        print("Too high! Try again.")
    elif guess < number:
        print("Too low! Try again.")
    else:
        print(f"🎉 Congratulations {name}! You guessed the number in {tries} tries!")
        break
```

```
Enter your name: esha
Welcome esha! Let's play a number guessing game.
Enter your guess (1-100): 50
Too low! Try again.
Enter your guess (1-100): 40
Too low! Try again.
Enter your guess (1-100):
```