SkillShikshya

Learn skills that matter

Python and Django

Presented by: Prashant Karna

Puzzle 1

Prisoners and Hats Puzzle:

There are 10 prisoners standing in a line. Each prisoner can see the hats of all the prisoners in front of them but cannot see their own hat or the hats of those behind them. Each prisoner is wearing a hat that is either black or white. The prisoners are not allowed to communicate with each other once the line is formed but are allowed to strategize beforehand.

The rules are as follows:

1. **Starting from the back of the line (the prisoner who sees all the others), each prisoner must call out a color: "black" or "white".**
2. **If a prisoner guesses the color of their own hat correctly, they are considered safe; otherwise, they are eliminated.**
3. **The goal is to maximize the number of prisoners saved**.

Question:

What strategy should the prisoners agree upon to save the maximum number of prisoners, and how many can they guarantee to save?

Puzzle 1

Prisoners and Hats Puzzle Solution (BWWBWBWWBW)

Strategy:

Prisoners agree to use parity (odd/even) of black hats as a reference.

1. Prisoner 1 (at the back) announces the parity of black hats they see:
    - Says "**Black**" if the count is **odd**, "**White**" if **even**.
    - This sets the reference for all others.
2. Each subsequent prisoner:
    - Counts black hats they see.
    - Deduces their own hat color to maintain the announced parity.

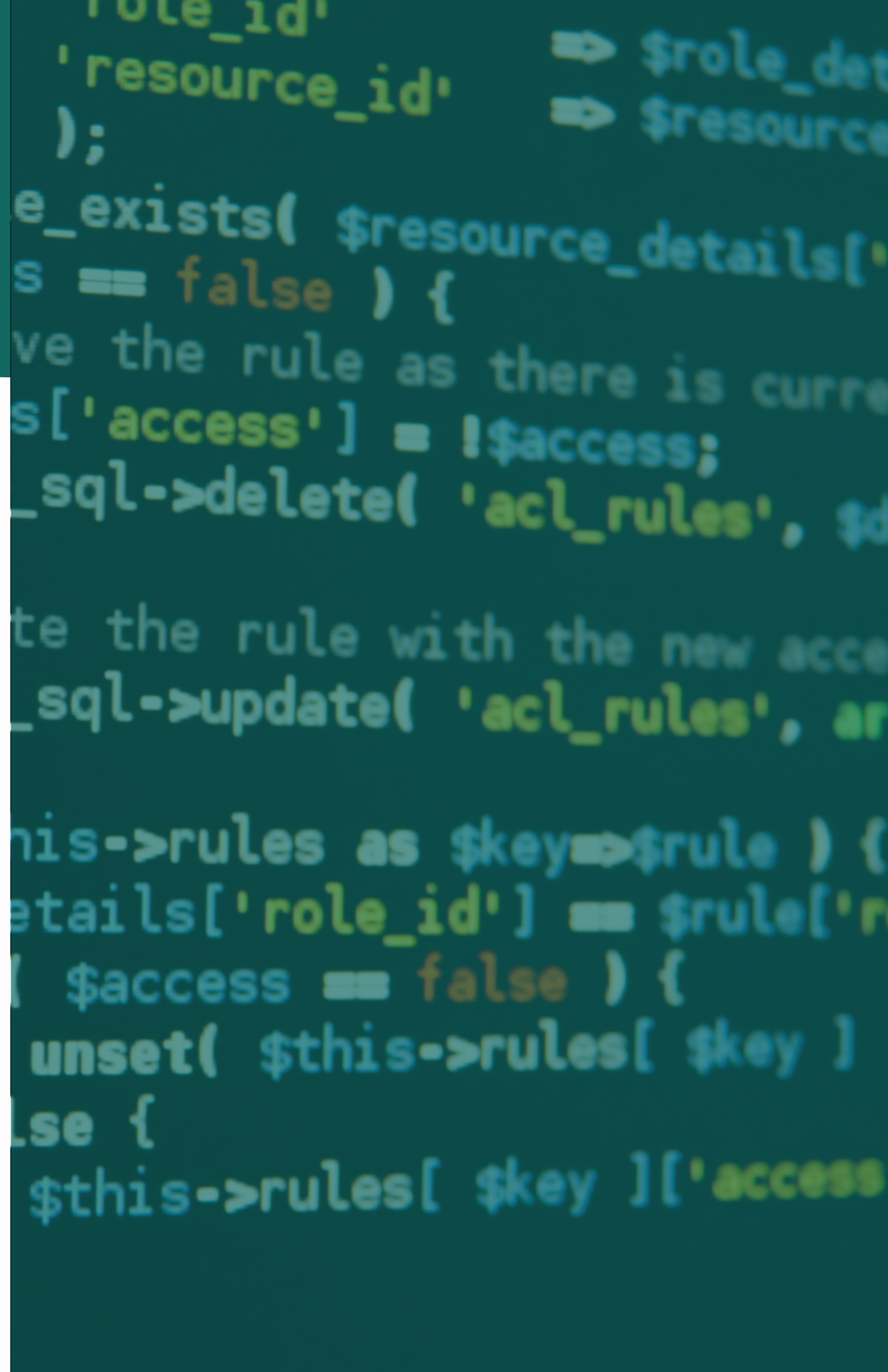
Walkthrough:

- Prisoner 1: Sees 4 black hats (even), says "White".
- Prisoner 2: Sees 4 black hats, deduces own hat is "White".
- Continue similarly; each prisoner adjusts based on parity.

Guarantee:

- Prisoner 1 has a 50% chance of guessing correctly.
- 9 out of 10 prisoners are guaranteed to survive.

Table Of Contents



SkillShikshya

Learn skills that matter

Version Control System

- **What is Version Control?**
  - A system that records changes to files over time, allowing you to manage and track the history of changes.
- **Key Benefits:**
  - **Collaboration**: Multiple developers can work on the same project without overwriting each other's changes.
  - **History Tracking:** You can track changes, roll back to previous versions, and resolve conflicts.
  - **Backup:** VCS provides a safe backup for your project files, storing multiple versions of the same file.

SkillShikshya
Learn skills that matter

Version Control System

- **Git:**
- **Definition:** A distributed version control system (VCS) that tracks changes in files and allows multiple developers to collaborate on code.
- **Functionality:**
  - Manages and records changes to files locally on a developer's machine.
  - Can be used without any internet connection.
  - Provides tools for branching, merging, and managing different versions of code.
- **Use Case:** Tracks code changes, helps in collaboration on local repositories, version history, and branching.

SkillShikshya

Learn skills that matter

Version Control System

- **GitHub:**
- **Definition:** A cloud-based platform for hosting Git repositories, enabling collaboration through features like pull requests, issues, and project boards.
- **Functionality:**
  - Hosts Git repositories remotely on the cloud.
  - Allows collaboration between developers by sharing repositories, reviewing pull requests, and tracking project tasks.
  - Provides a web interface for managing repositories and issues.
- **Use Case:** Hosts projects, facilitates online collaboration, code sharing, and project management.

SkillShikshya

Learn skills that matter

Git Commands

- **git init –** Initialize a new Git repository
- **git clone <repository-url> –** Clone a repository to your local machine
- **git status –** Show the status of working directory and staging area
- **git add <filename> –** Stage a specific file for commit
- **git commit -m "message" –** Commit staged changes with a message
- **git log –** View the commit history
- **git branch <branch-name> –** Create a new branch
- **git checkout <branch-name> –** Switch to a different branch
- **git checkout -b <branch-name> –** Create and switch to a new branch
- **git merge <branch-name> –** Merge another branch into the current one
- **git push –** Push changes to the remote repository
- **git pull –** Fetch and merge changes from the remote repository
- **git reset <file> –** Unstage a file
- **git checkout -- <file> –** Discard changes in a file

EOD 1

SkillShikshya

Learn skills that matter

Compiler Vs Interpreter

Difference Between Compiler and Interpreter

A **compiler** and an **interpreter** are both used to translate high-level programming languages (like Python, Java, and C++) into machine code, but they work differently.

Feature	Compiler	Interpreter
Execution	Translates the entire program at once before execution.	Translates and executes the program **line-by-line**.
Speed	Faster because it generates an optimized executable file.	Slower since it translates code on the fly.
Error Handling	Displays all errors **at once** after compilation.	Stops at the first error, making debugging easier but slower.
Usage	Used in languages like **C, C++, Java** (before execution).	Used in languages like **Python, JavaScript, Ruby**.
Output	Produces a separate executable file (.exe, .out, etc.).	Does not produce a separate file; runs directly.
Example Compiler	GCC (for C, C++), Java Compiler (Javac).	Python Interpreter (CPython), Node.js (for JavaScript).

V/S

1. Install Python
Download Python: Go to Python's official download page [*https://www.python.org/downloads/*] and choose the latest version for your operating system (Windows, Mac, or Linux).

Windows: Run the installer, and be sure to check "Add Python to PATH" during installation.
Mac: Download and run the installer package.
Linux: Most distributions come with Python pre-installed, but you can download it from the Python site if needed.

2. Install Django

Create a Virtual Environment:
- `python3 -m venv myenv`

Activate the Virtual Environment:
- `Mac/Linux: source myenv/bin/activate`
- `Windows: myenv\Scripts\activate`

Install Django:
- `pip install django`

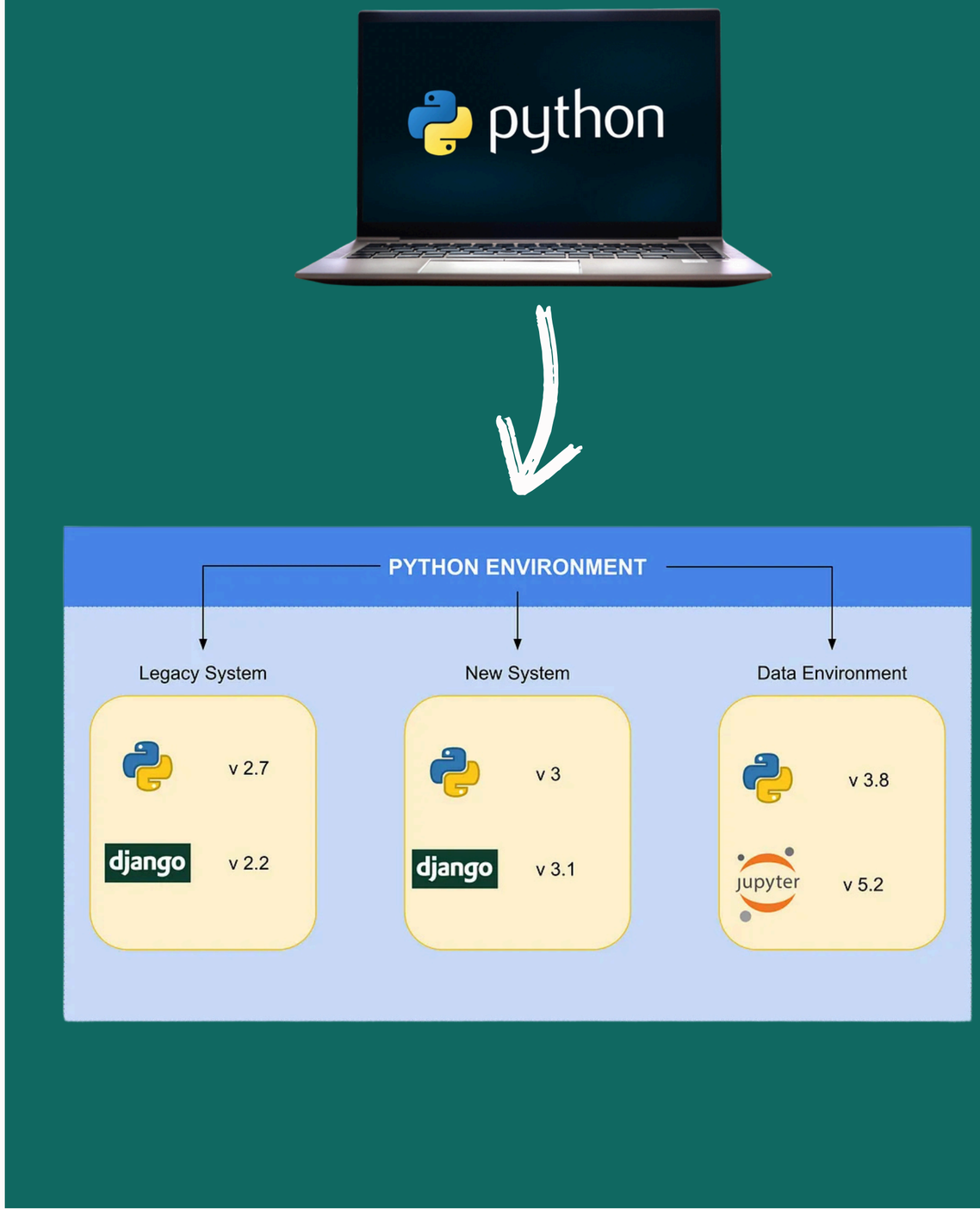
Confirm Django Installation:
- `django-admin --version`

Installation of Python Django

SkillShikshya
Learn skills that matter

Virtual Environment

- **Why Use It?**
  - Isolates project dependencies
  - Prevents version conflicts
  - Keeps the global setup clean
- ⚙️ **Quick Setup:**
  - **Install**: pip install virtualenv
  - **Create**: python -m venv myenv
  - **Activate**:
  - ***Windows***: myenv\Scripts\activate
  - ***Mac/Linux*:** source myenv/bin/activate
  - **Install Django**: pip install django
  - **Deactivate**: deactivate



Python Roadmap

Step-by-step guide to mastering Python, from basics to building real-world applications.

SkillShikshya
Learn skills that matter

Step 1: Learn the Basics- Syntax, Variables, Data Types, Conditionals,

Step 2: Loops, Functions, Built-in Functions

Step 3: Data Structures- Lists, Tuples, Sets, Dictionaries

Step 4: OOP- Classes, Inheritance, Objects

Step 5: Advance Topics 1- RegEx, Decorators, Lambda

Step 6: Advanced Topics 2- Modules, Iterators,

Step 7: Learn Python Libraries

Step 8: Learn Version Control Systems

Step 9: Build Python Apps

6-Step Beginner’s Guide to Django

Python Basics

Master syntax, data types, loops, functions, and OOP concepts.

Django Basics

Set up Django, understand the Model-View-Template (MVT) structure, work with models, views, URLs, and templates.

Models & Database

Learn how to define models (database tables) and use Django's ORM to save and retrieve data.

Views & URLs

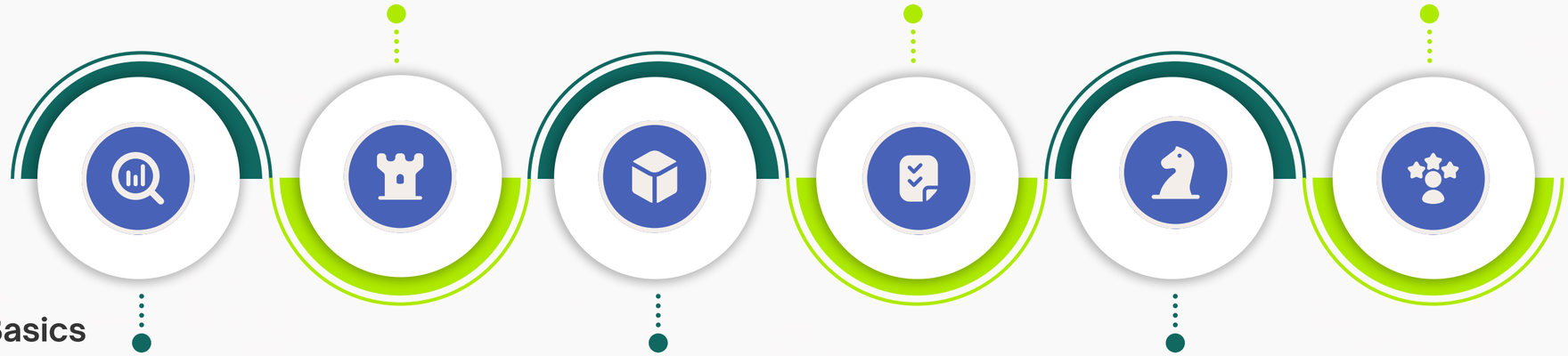
Create views to control what your app shows, and link them to URLs so users can access different pages.

Templates

Use Django templates to render HTML with dynamic data, making your web pages interactive.

Basic Forms & User Login

Add forms for data input, and implement basic login and registration.



SkillShikshya

Learn skills that matter

How to Be Logical in Programming

1. **Understand the Problem**: Break it into smaller parts; identify inputs/outputs.
2. **Plan Before Coding:** Use flowcharts or pseudocode to outline logic.
3. **Think Step-by-Step**: Solve sequentially, visualizing data flow.
4. **Learn Algorithms & Data Structures:** Master basics like sorting, recursion, and arrays.
5. **Debug Systematically:** Fix small chunks; trace issues with logs.
6. **Test Extensively:** Handle edge cases and invalid inputs.
7. **Simplify:** Avoid overthinking; break problems into manageable steps.
8. **Practice Regularly:** Solve puzzles, analyze code, and refine skills.

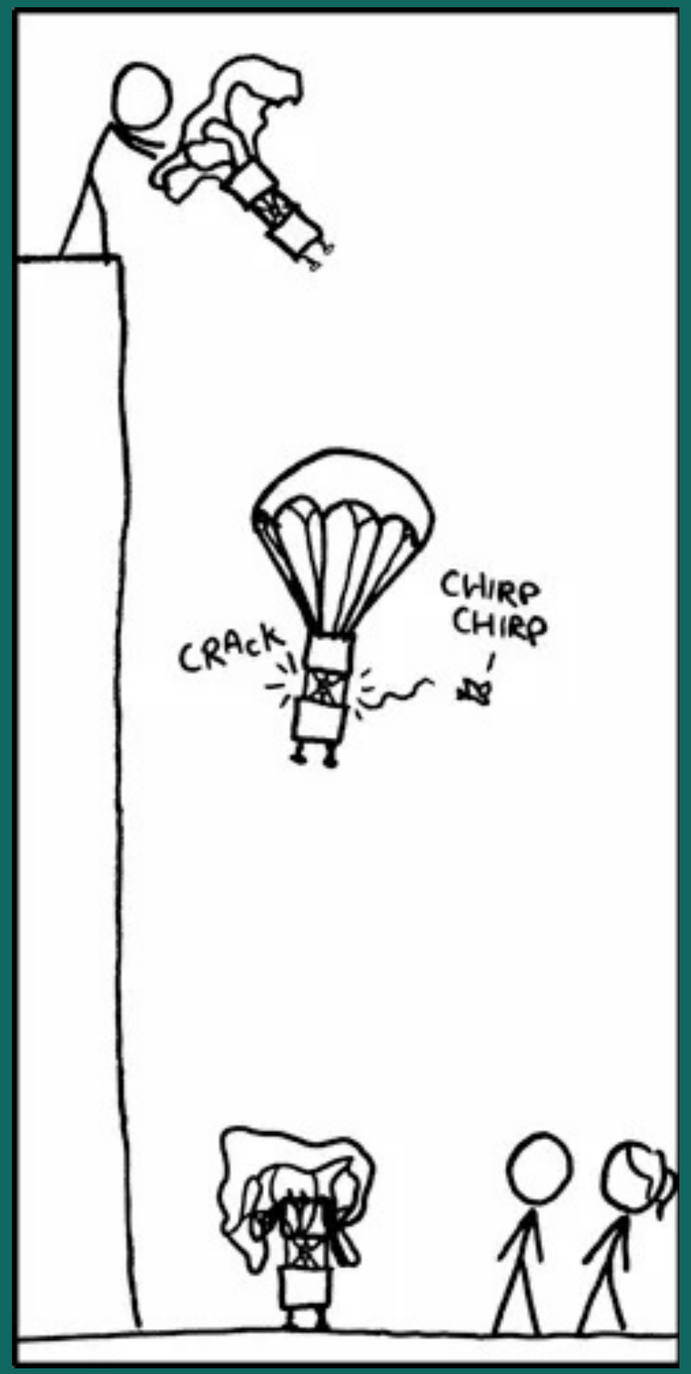
"Logic is the foundation of great programming!"

SkillShikshya

Learn skills that matter

Lets Solve one more Question

A building has **100 floors.** One of the floors is the highest floor an egg can be dropped from without breaking. If an egg is dropped from above that floor, it will break. If it is dropped from that floor or below, it will be completely undamaged and you can drop the egg again. Given two eggs, **find the highest floor an egg can be dropped from without breaking,** with as few drops as possible. approaches for optimal solution.



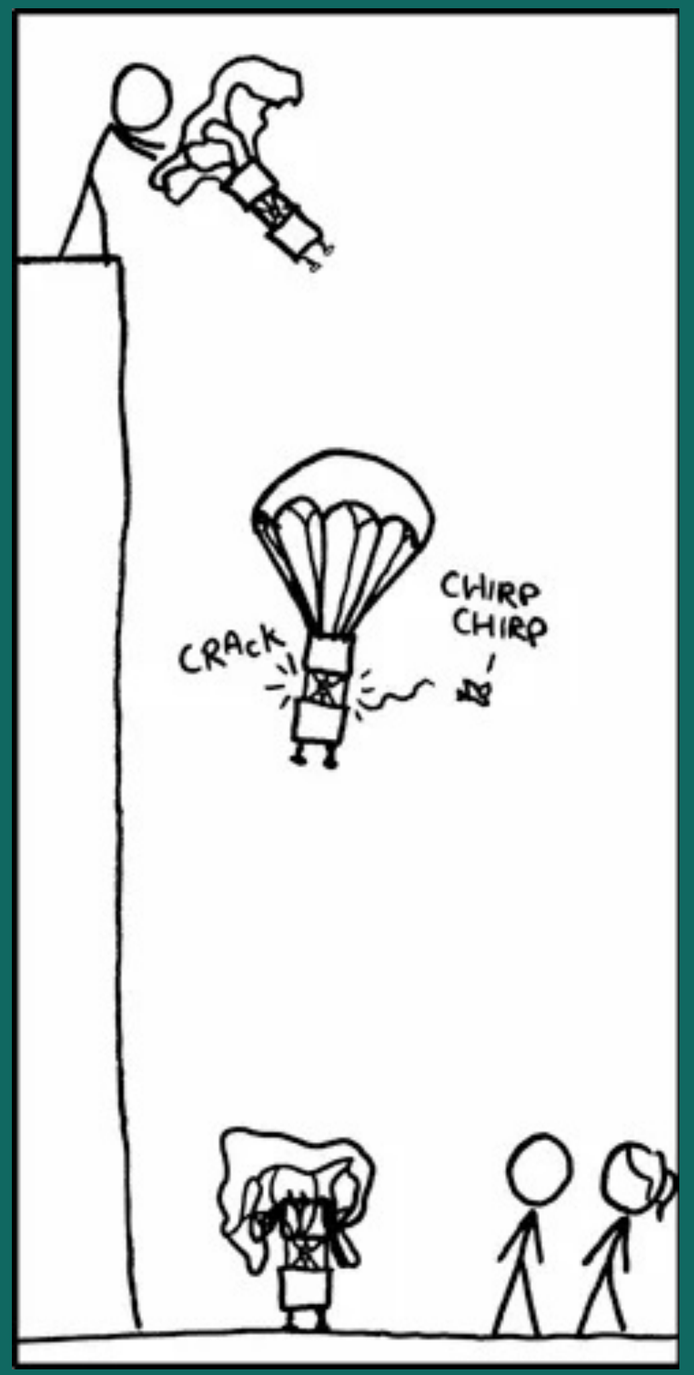
SkillShikshya

Learn skills that matter

Solution

Approach 1: ***Linear Search (Worst Case: 100 Drops)***

A naive approach would be to start from floor 1 and move upwards one floor at a time until the egg breaks. If the first egg breaks at floor f, then we know that the highest safe floor is f−1. This takes a worst-case of 100 drops (if the safe floor is 100).



SkillShikshya

Learn skills that matter

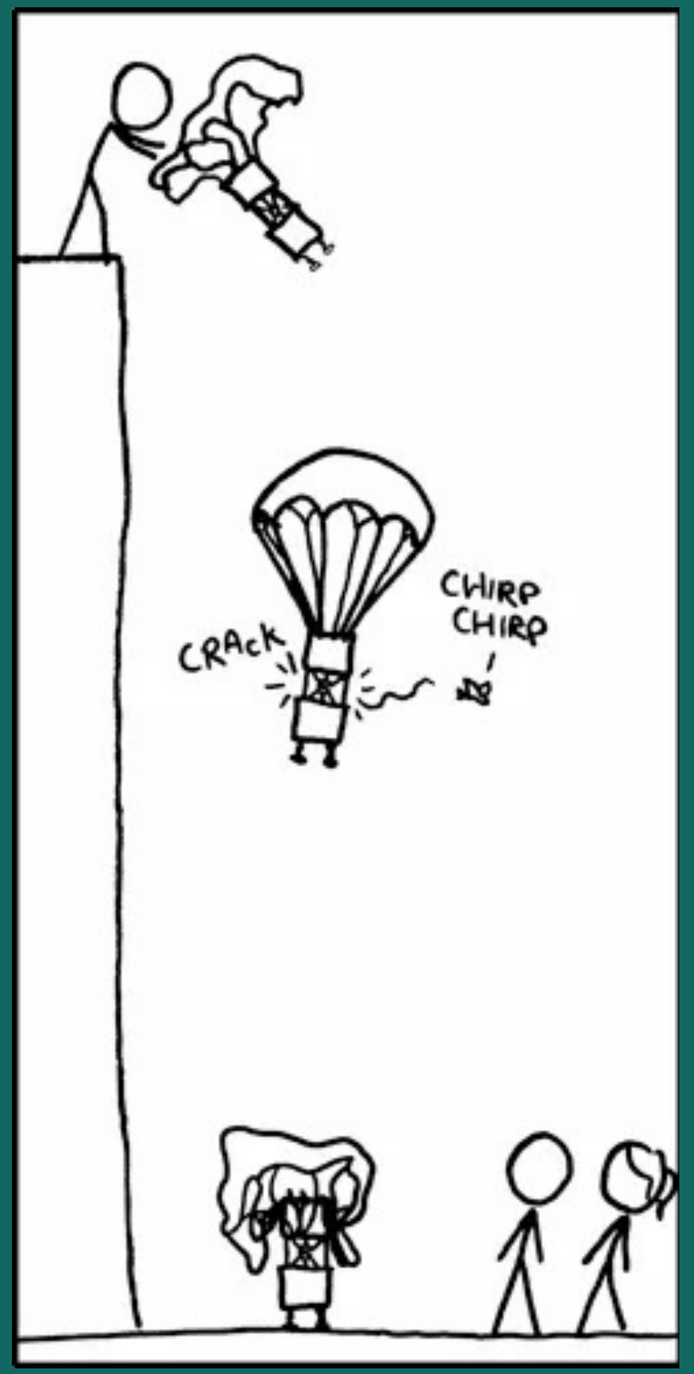
Solution

Approach 2: Binary Search (Worst Case: 50 Drops)

A common approach is binary search:

1. Drop the first egg from floor 50.
2. If it breaks, perform a linear search from floors 1 to 49 with the second egg.
3. If it doesn't break, drop the first egg from 75, then 87, etc., halving the range each time.

- **Worst-case:** 50 drops (if the second egg has to do a full linear scan).



SkillShikshya

Learn skills that matter

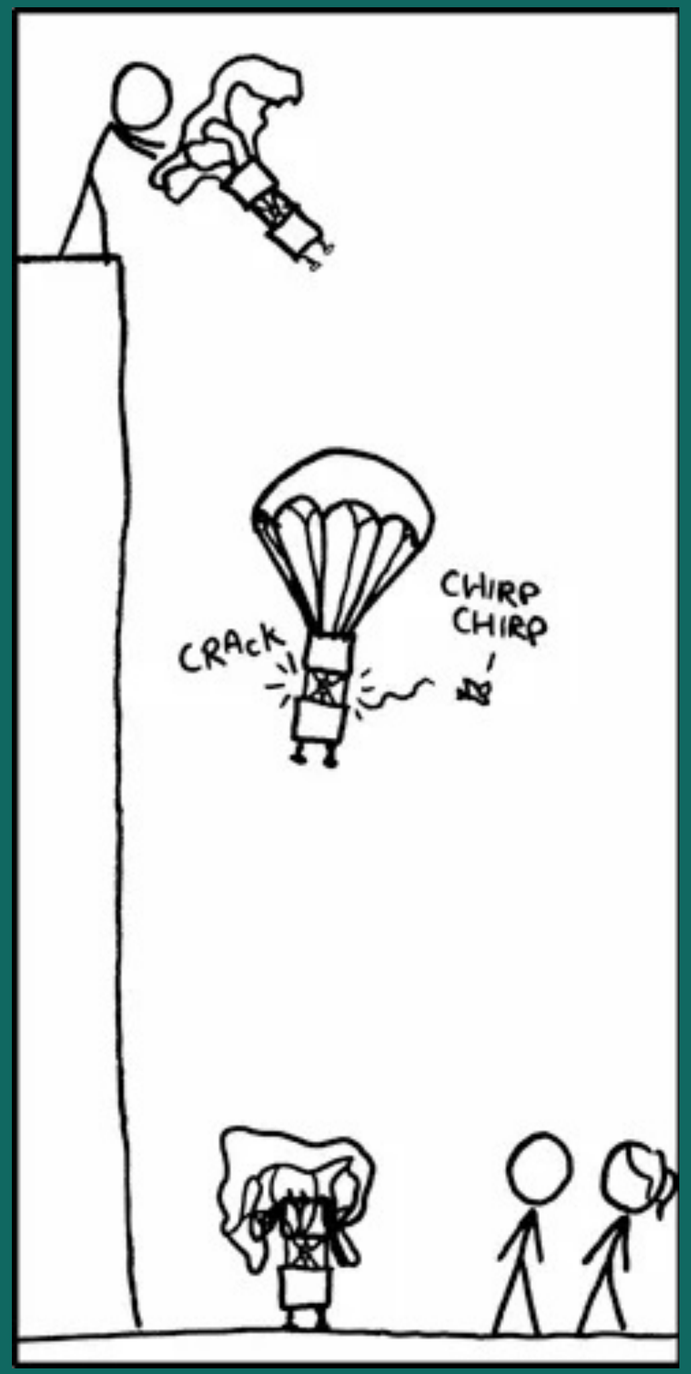
Solution

Approach 3: Optimized Two-Egg Strategy (Worst Case: 14 Drops)

Instead of using a binary search, we use a mathematical approach that minimizes the worst-case number of drops:

Strategy

1. Drop the first egg at increasing intervals so that the worst-case number of drops is minimized.
2. If the first egg breaks at some floor xxx, use the second egg to search linearly from the previous safe floor.



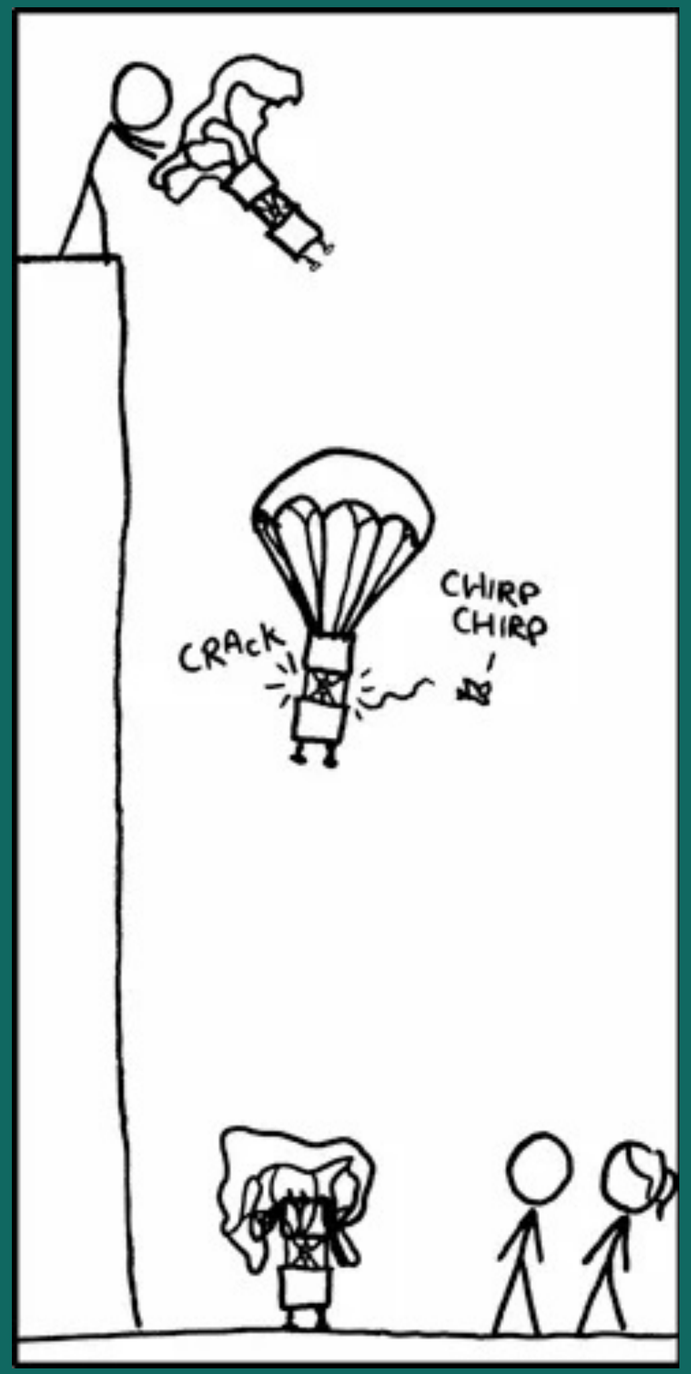
SkillShikshya

Learn skills that matter

Solution

The optimal interval to drop the first egg starts from floor 14, then increases by one less than the previous jump:

- Drop from floor 14 → If it breaks, check floors 1-13 linearly.
- Else, drop from floor 27 (14 + 13) → If it breaks, check floors 15-26.
- Else, drop from floor 39 (27 + 12) → If it breaks, check floors 28-38.
- Else, drop from floor 50 (39 + 11).
- Else, drop from floor 60 (50 + 10).
- Else, drop from floor 69 (60 + 9).
- Else, drop from floor 77 (69 + 8).
- Else, drop from floor 84 (77 + 7).
- Else, drop from floor 90 (84 + 6).
- Else, drop from floor 95 (90 + 5).
- Else, drop from floor 99 (95 + 4).
- Else, drop from floor 100 (99 + 1).



SkillShikshya

Learn skills that matter

Solution

Why This Works

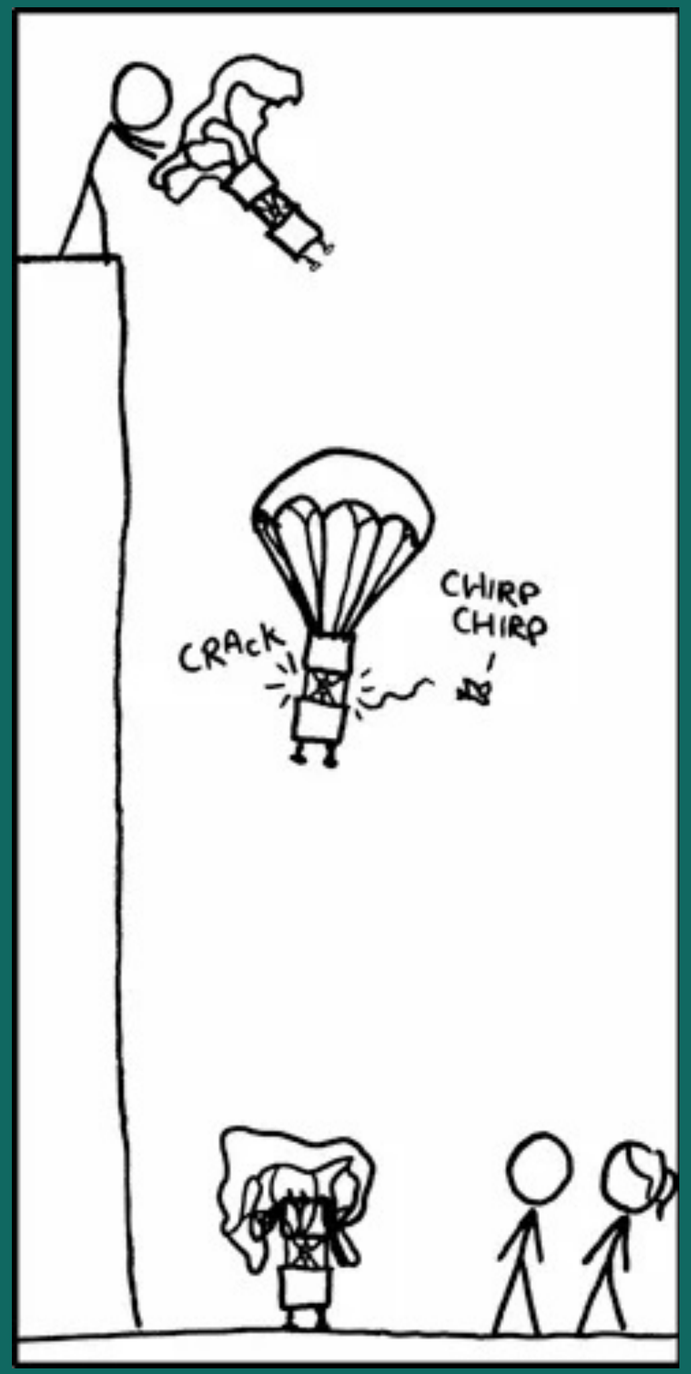
- The first egg will break at some floor x, and the second egg will check the remaining floors sequentially.
- This approach ensures that the maximum number of drops is at most 14 in the worst case.

Conclusion

The optimal way to solve the problem with two eggs is to drop the first egg in increasing intervals (1+2+3+...+n ≥ 100). This gives a worst-case of 14 drops, which is the best we can do.

Formula:

$x(x + 1)/2 = 100$



EOD 2

SkillShikshya

Learn skills that matter

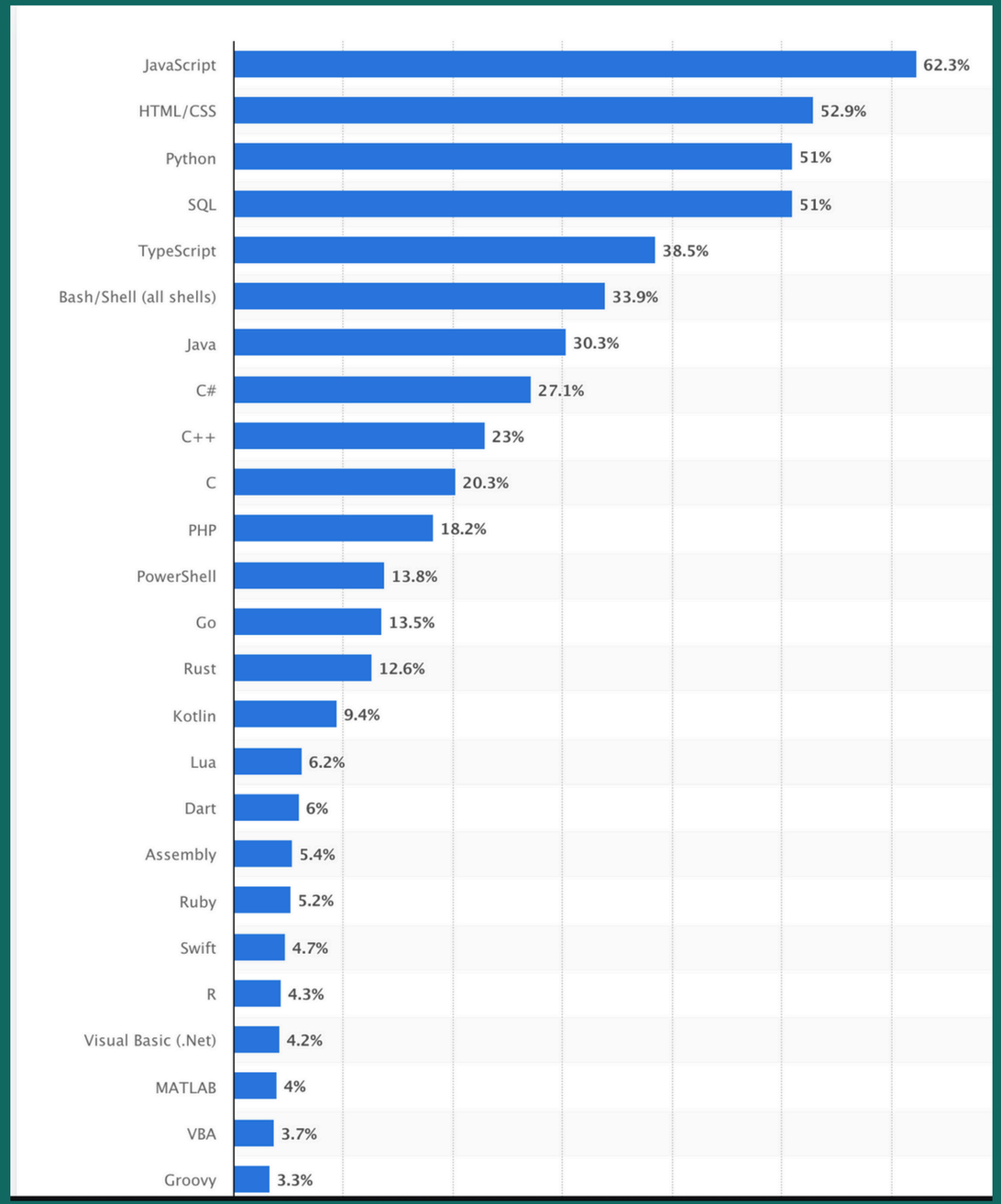
Introduction

What is Python?

- High-level, versatile programming language.
- Created by Guido van Rossum in the late 1980s.
- Used in web development, data science, automation, and AI.

Why Choose Python?

- Easy-to-read, English-like syntax.
- Cross-platform and versatile.
- Extensive libraries (e.g., NumPy, Pandas).
- Strong community support.
- Supports multiple programming paradigms.

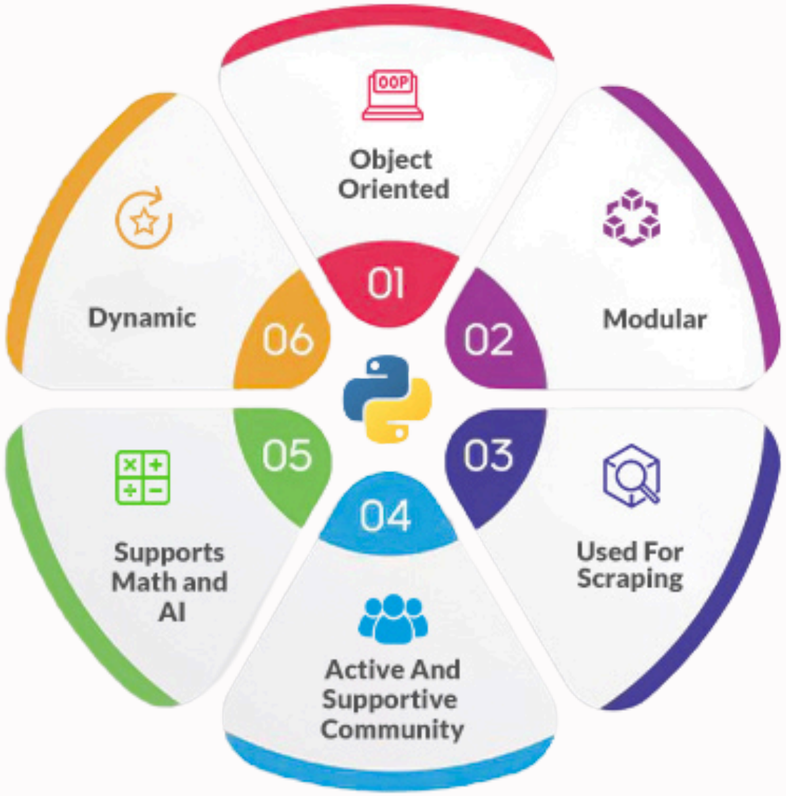


SkillShikshya
Learn skills that matter

Introduction

Why Use Python Despite Being Slower?

- Boosts productivity with quick development.
- Ideal for rapid prototyping.
- Optimization tools available (e.g., Cython).
- Large ecosystem outweighs speed drawbacks.



AREN'T YOU ALL WORRIED ABOUT RECENT GOOGLE LAYOFFS OF PYTHON DEVELOPER?

SkillShikshya

Learn skills that matter

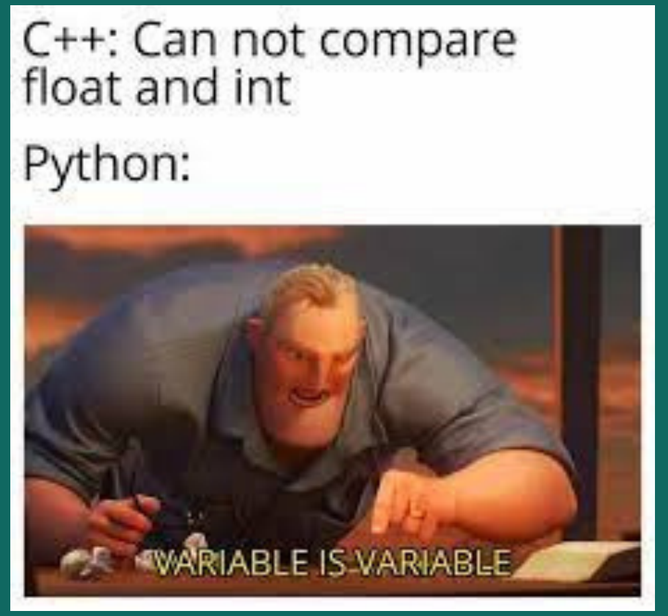
Introduction to Python Syntax

- Simple and Readable: Python code is designed to be clean and intuitive, resembling plain English.
- Indentation: Uses indentation (whitespace) to define code blocks instead of braces {}. Ensures readability and consistency.
- Variables: No need to declare variable types; Python infers them automatically.

```python
def greet_user(name):
    message = "Hello, " + name + "!"
    return message

# Main code
user_name = "Alice"
greeting = greet_user(user_name)

print(greeting)
```



SkillShikshya

Learn skills that matter

Introduction to Python Syntax

- **Basic Data Types:** Common types include int, float, str, and bool.
- **Control Structures:**
  i. if, elif, else for conditions.
  ii. for and while loops for iteration.
  if age > 18:
  print("Adult")
- **Functions:** Defined using def keyword.
  def greet(name):
  return f"Hello, {name}!"

Python syntax is designed for clarity, making it easy to learn and write clean, efficient code.

```python
# Basic Data Types
age = 21            # int
height = 5.9        # float
name = "Alice"      # str
is_student = True   # bool

# Control Structures

# Conditionals
if age > 18:
    print("Adult")
elif age == 18:
    print("Just became an adult")
else:
    print("Minor")

# For loop
for i in range(5):  # Iterates from 0 to 4
    print(f"Number: {i}")

# While loop
count = 0
while count < 3:
    print("Counting:", count)
    count += 1

# Functions
def greet(name):
    return f"Hello, {name}!"

# Function call
print(greet(name))  # Output: Hello, Alice!

```

SkillShikshya

Learn skills that matter

Introduction to Variables in Python

📌 **What is a Variable?**

- A variable is a named storage location for data in memory.
- It allows storing, modifying, and reusing values in a program.

🔹 **Example:**

- name = "Alice"
- age = 25
- price = 99.99

💡 **Tip: Variables help make programs dynamic and flexible.**

```python
# Basic Data Types

# int: Whole numbers (positive or negative)
age = 25
temperature = -5

# float: Decimal numbers
height = 5.9
weight = -3.14

# str: Text data, enclosed in quotes
greeting = "Hello, World!"

# bool: True or False values
is_active = True
is_verified = False

# Complex Data Types

# list: Ordered, mutable collection
fruits = ["apple", "banana", "cherry"]

# tuple: Ordered, immutable collection
coordinates = (10, 20)

# dict: Key-value pairs, like a map
person = {"name": "Alice", "age": 25}

# set: Unordered collection with unique items
unique_numbers = {1, 2, 3, 4}

# Printing each data type to see the output
print("Age:", age)
print("Height:", height)
print("Greeting:", greeting)
print("Is Active:", is_active)
print("Fruits:", fruits)
print("Coordinates:", coordinates)
print("Person:", person)
print("Unique Numbers:", unique_numbers)

```

SkillShikshya

Learn skills that matter

Variable Naming Rules & Conventions

📝 **Rules for Naming Variables:**

- Must start with a letter (A-Z or a-z) or an underscore (_)
- Cannot start with a number (e.g., 2name ❌)
- Can contain letters, numbers, and underscores
- Case-sensitive (Age and age are different)
- Avoid using Python keywords (e.g., class, def, if)

✅ **Examples:**

- Valid: student_name, _count, price1
- Invalid: 1student, class, name@

💡 **Tip: Use meaningful variable names to improve code readability.**

```python
# Basic Data Types

# int: Whole numbers (positive or negative)
age = 25
temperature = -5

# float: Decimal numbers
height = 5.9
weight = -3.14

# str: Text data, enclosed in quotes
greeting = "Hello, World!"

# bool: True or False values
is_active = True
is_verified = False

# Complex Data Types

# list: Ordered, mutable collection
fruits = ["apple", "banana", "cherry"]

# tuple: Ordered, immutable collection
coordinates = (10, 20)

# dict: Key-value pairs, like a map
person = {"name": "Alice", "age": 25}

# set: Unordered collection with unique items
unique_numbers = {1, 2, 3, 4}

# Printing each data type to see the output
print("Age:", age)
print("Height:", height)
print("Greeting:", greeting)
print("Is Active:", is_active)
print("Fruits:", fruits)
print("Coordinates:", coordinates)
print("Person:", person)
print("Unique Numbers:", unique_numbers)

```

SkillShikshya

Learn skills that matter

Types of Variables in Python

📊 **Common Data Types:**

- *Integer (int)*: age = 30
- *Floating Point (float)*: height = 5.8
- *String (str)*: city = "New York"
- *Boolean (bool)*: is_active = True
- *List:* fruits = ["apple", "banana", "cherry"]
- *Tuple*: coordinates = (10, 20)
- *Dictionary*: person = {"name": "Alice", "age": 25}

💡 **Tip: Use type(variable_name) to check the type of a variable.**

```python
# Basic Data Types

# int: Whole numbers (positive or negative)
age = 25
temperature = -5

# float: Decimal numbers
height = 5.9
weight = -3.14

# str: Text data, enclosed in quotes
greeting = "Hello, World!"

# bool: True or False values
is_active = True
is_verified = False

# Complex Data Types

# list: Ordered, mutable collection
fruits = ["apple", "banana", "cherry"]

# tuple: Ordered, immutable collection
coordinates = (10, 20)

# dict: Key-value pairs, like a map
person = {"name": "Alice", "age": 25}

# set: Unordered collection with unique items
unique_numbers = {1, 2, 3, 4}

# Printing each data type to see the output
print("Age:", age)
print("Height:", height)
print("Greeting:", greeting)
print("Is Active:", is_active)
print("Fruits:", fruits)
print("Coordinates:", coordinates)
print("Person:", person)
print("Unique Numbers:", unique_numbers)

```

SkillShikshya

Learn skills that matter

Variables and Data Types

📌 What is Variable Scope?

- Scope defines where a variable can be accessed within a program.

🔹 Types of Variable Scope:

- ***Local Scope*** (Declared inside a function, accessible only there)
- ***Global Scope*** (Declared outside functions, accessible everywhere)
- **Enclosed Scope** (Variables inside nested functions)
- **Built-in Scope** (Predefined Python functions like print(), len())

💡 **Tip: Use the global keyword inside functions to modify global variables.**

```python
# Basic Data Types

# int: Whole numbers (positive or negative)
age = 25
temperature = -5

# float: Decimal numbers
height = 5.9
weight = -3.14

# str: Text data, enclosed in quotes
greeting = "Hello, World!"

# bool: True or False values
is_active = True
is_verified = False

# Complex Data Types

# list: Ordered, mutable collection
fruits = ["apple", "banana", "cherry"]

# tuple: Ordered, immutable collection
coordinates = (10, 20)

# dict: Key-value pairs, like a map
person = {"name": "Alice", "age": 25}

# set: Unordered collection with unique items
unique_numbers = {1, 2, 3, 4}

# Printing each data type to see the output
print("Age:", age)
print("Height:", height)
print("Greeting:", greeting)
print("Is Active:", is_active)
print("Fruits:", fruits)
print("Coordinates:", coordinates)
print("Person:", person)
print("Unique Numbers:", unique_numbers)

```

SkillShikshya

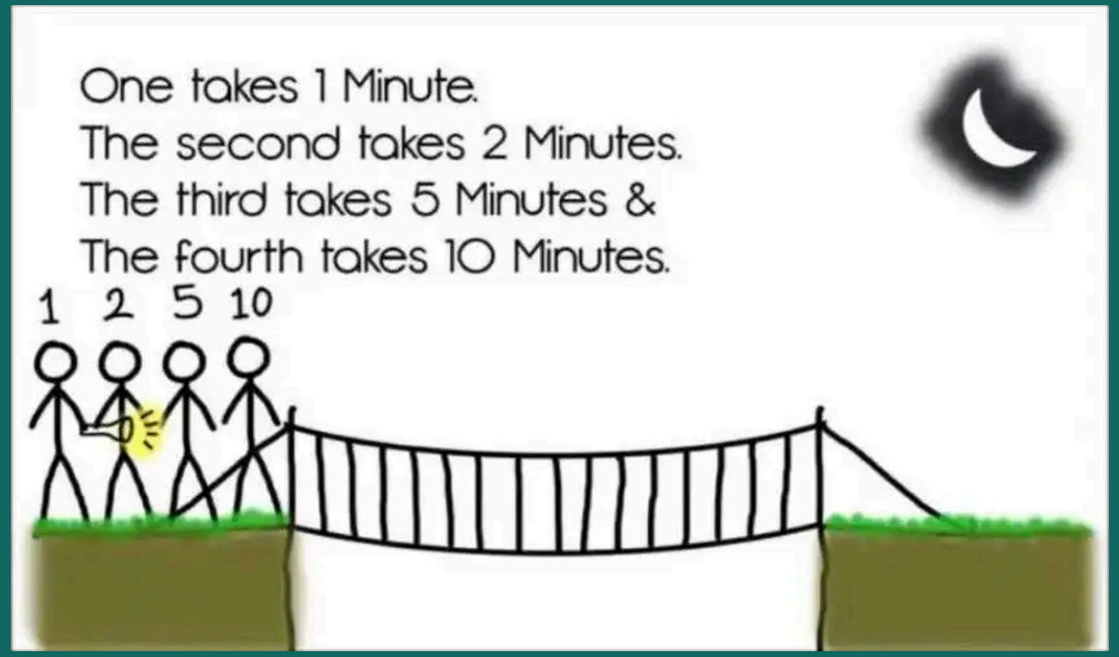
Learn skills that matter

Puzzle Time

Four people need to cross a bridge. It's nighttime and pretty dark. There's only one flashlight; it's dangerous to cross the bridge without one. The bridge can only support two people at a time. Each person will take a different amount of time to cross the bridge: 1 min, 2 mins, 5 mins, and 10 mins. What is the shortest possible time for all four people to cross the bridge?

NOTE: A bridge will collapse in 17 minutes.

This puzzle is designed to test your ability to come up with the most efficient way to solve a problem, especially with constraints. This is an important skill to have for a software developer where solutions need to be as efficient as possible.

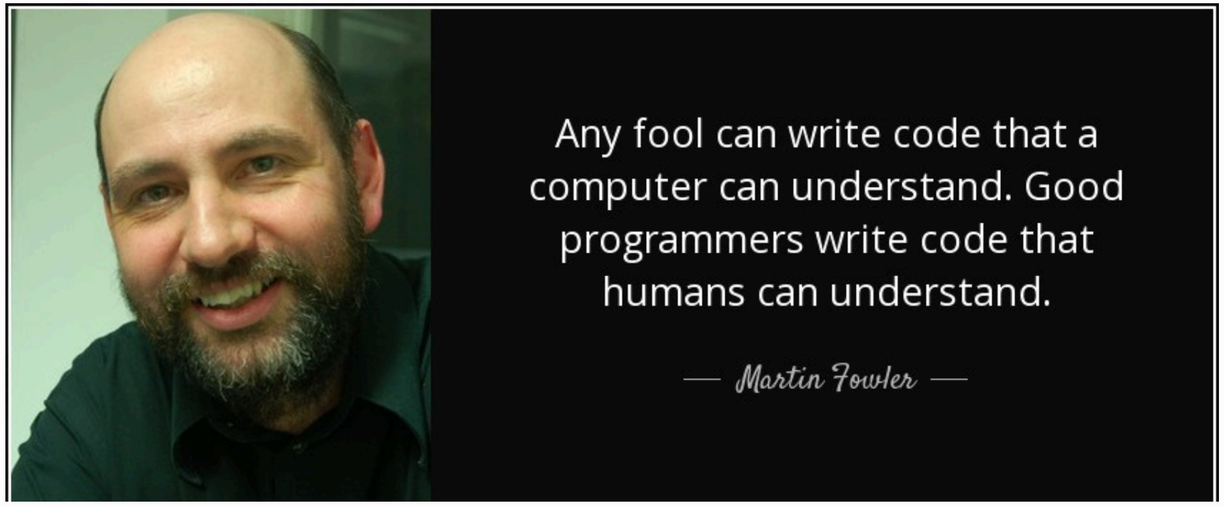


EOD 3

SkillShikshya

Learn skills that matter

Open your IDE and lets start few coding...




```python
#Local Variable
def my_function():
    x = 10  # Local variable
    print(x)
my_function()
print(x)  # ❌ Error: x is not accessible outside the function

```

```python
#Global Variable
y = 20  # Global variable
def my_function():
    print(y)  # ✅ Accessible inside the function
my_function()

```

```python
#Mutable Data Types
my_list = [1, 2, 3]
my_list.append(4)  # ✅ Modifies the existing list

my_list = [1, 2, 3]
del my_list[1]  # Removes element at index 1

```

SkillShikshya

Learn skills that matter

Operators in Python

📌 **What are Operators?**

- Operators perform operations on variables and values.
- Python supports different types of operators for calculations, comparisons, and logic.

🔷 **Types of Operators:**

1. **Arithmetic Operators** – Perform mathematical operations
2. **Comparison Operators** – Compare values
3. **Logical Operators** – Combine boolean expressions
4. **Assignment Operators** – Assign and modify variables
5. **Bitwise Operators** – Work with binary numbers
6. **Membership & Identity Operators** – Check existence and identity

💡 **Tip: Operators make expressions powerful and dynamic.**

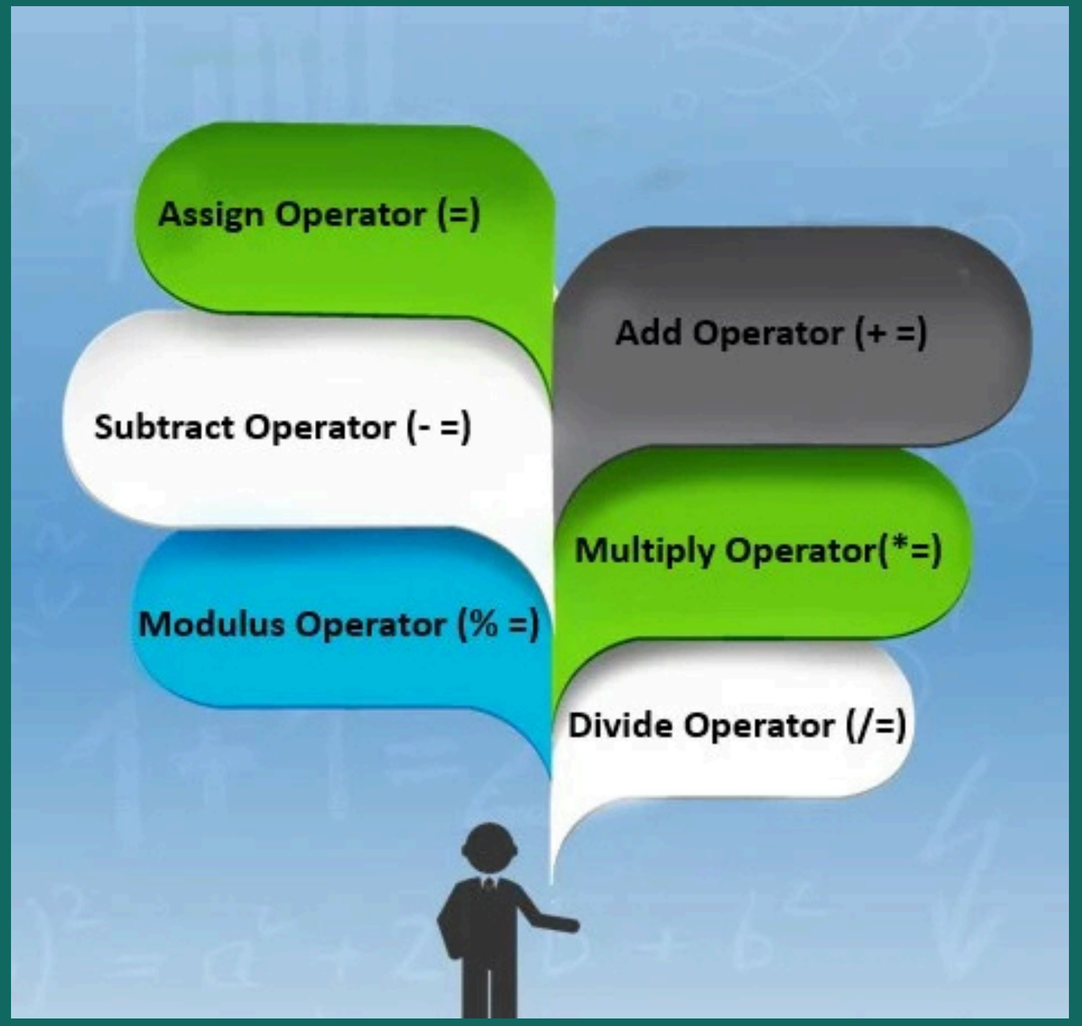
SkillShikshya

Learn skills that matter

Assignment Operators

📌 Modify and Assign Values

Operator	Example (`x = 10`)	Equivalent To	Result
`=`	`x = 10`	Assigns `10`	`10`
`+=`	`x += 5`	`x = x + 5`	`15`
`-=`	`x -= 3`	`x = x - 3`	`7`
`*=`	`x *= 2`	`x = x * 2`	`20`
`/=`	`x /= 2`	`x = x / 2`	`5.0`
`//=`	`x //= 3`	`x = x // 3`	`3`
`%=`	`x %= 3`	`x = x % 3`	`1`
`**=`	`x **= 2`	`x = x ** 2`	`100`



SkillShikshya

Learn skills that matter

Logical Operators in Python

Operator	Meaning	Example (`x = True`, `y = False`)	Output
`and`	Both True	`x and y`	`False`
`or`	At least one True	`x or y`	`True`
`not`	Reverse value	`not x`	`False`

💡 **Tip:**

- Use **and** for strict conditions.
- Use **or** for flexible conditions.
- Use **not** to negate a condition.

```python
has_passport = False
has_visa = True

if has_passport or has_visa:
    print("Can travel abroad")  # ✅ Output: Can travel abroad
```

```python
age = 25
income = 50000

if age > 18 and income > 30000:
    print("Eligible for loan")  # ✅ Output: Eligible for loan
```

```python
is_raining = False

if not is_raining:
    print("Go outside!")  # ✅ Output: Go outside!
```

SkillShikshya

Learn skills that matter

Introduction to Functions

🔹 What is a Function?

A function is a reusable block of code designed to perform a specific task. It helps in making programs more modular, readable, and efficient.

🔹 Why Use Functions?

✔ Code Reusability – Write once, use multiple times
✔ Modularity – Organize complex programs into smaller parts
✔ Readability – Makes the code easier to understand
✔ Debugging – Helps in identifying and fixing errors

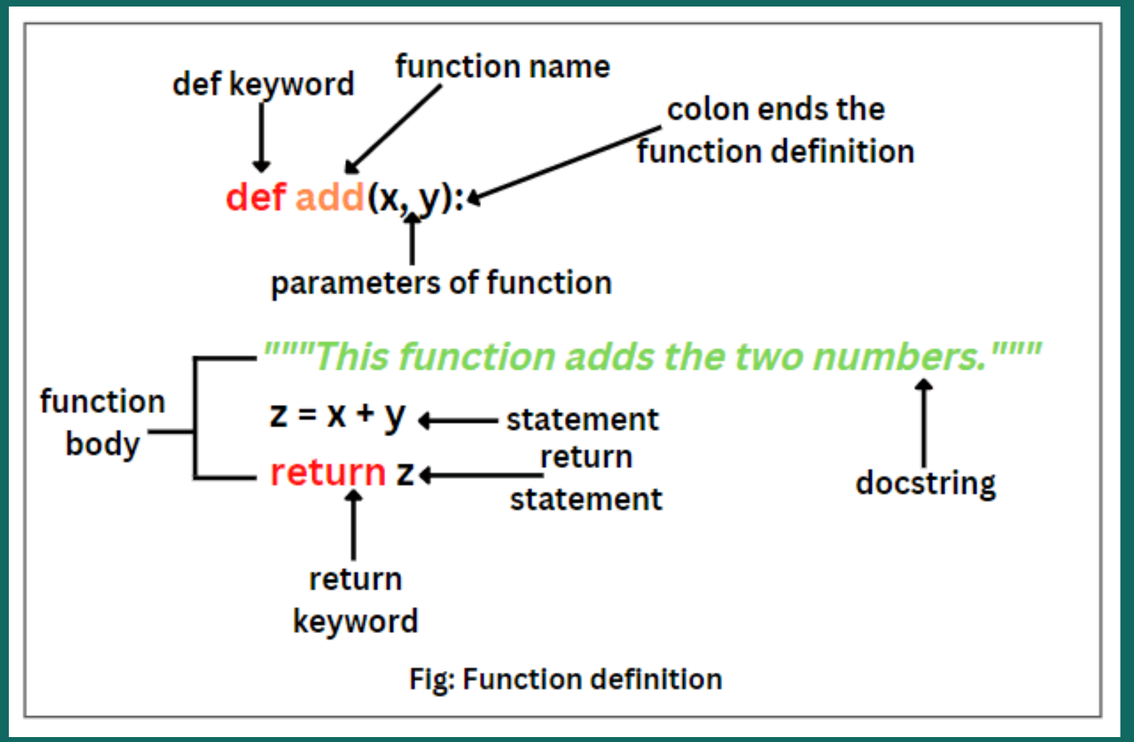
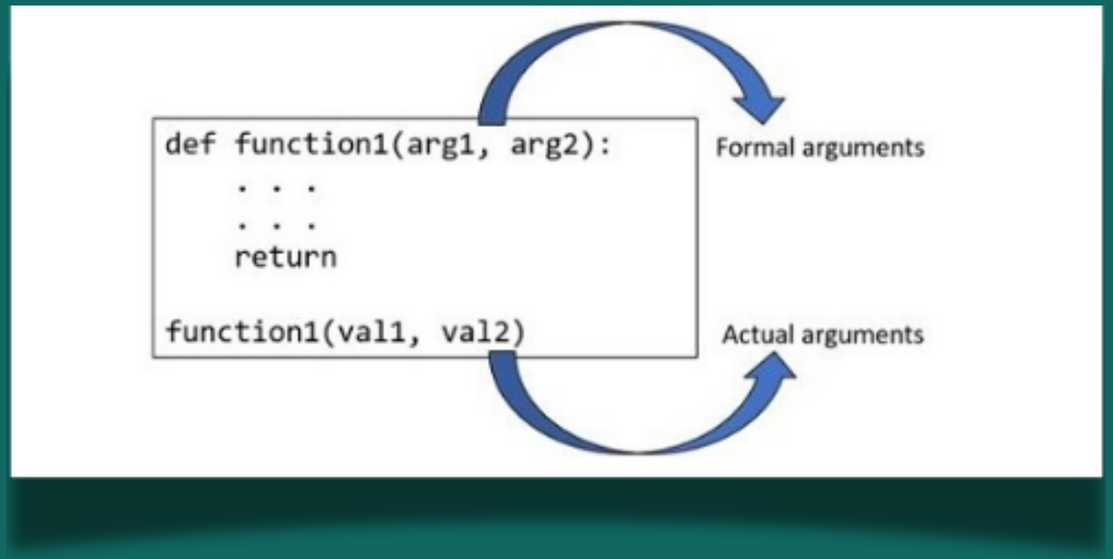


Fig: Function definition



SkillShikshya

Learn skills that matter

Types of Functions in Python 👀

Basic User-defined Function

Function with Parameters

Lambda Function (Short Anonymous Function)

Recursive Function (Function Calling Itself)

```python
def greet(name):
    return f"Hello, {name}!"

print(greet("Bibek"))
```

```python
def add_numbers(a, b):
    return a + b

result = add_numbers(5, 10)
print(result)
```

```python
square = lambda x: x * x
print(square(4))
```

```python
def factorial(n):
    if n == 1:
        return 1
    return n * factorial(n - 1)

print(factorial(5))
```

SkillShikshya
Learn skills that matter

Puzzle Time

Try to solve this:

Ajay and Vijay undertake to do a piece of work for Rs. 480. Ajay alone can do it in 75 days while Vijay alone can do it in 40 days. With the help of Pradeep, they finish the work in 25 days. How much should Pradeep get for his work?

SkillShikshya
Learn skills that matter

Puzzle Time

Answer:

In 24 days, they would have done 1/3 and 5/8 of the work.

The remaining work is 1 - (1/3 + 5/8) = 1/24. This means Pradeep has done 1/24th of the work, so he should be paid 1/24th of the amount

i.e. 480 × 1/24 = Rs. 20.

EOD 4