# PYTHON ESSENTIALS

Rudram Vyas

# Introduction

- This presentation covers intermediate Python concepts.
- We'll explore data structures, control flow, and modular programming.
- Each topic includes examples.
- We will be covering
  - Sets & Dictionaries
  - Looping Techniques [6 techniques ]
  - More on Conditions
  - Comparing Sequences and Other Types
  - Modules & Execution

# Sets

**What is a Set?**

A set is an unordered collection of **unique** elements.

Defined using curly braces {} or the set() constructor.

Mutable and no duplicate items allowed.

# Set Operations

- Add element: *s.add(x)*
- Remove element: *s.remove(x)* or *s.discard(x)*
- Membership test: x in s
- Mathematical operations:
  - Union: s1 | s2
  - Intersection: s1 & s2
  - Difference: s1 - s2
  - Symmetric difference (XOR): s1 ^ s2

# Dictionary

**Key Concepts of Dict**

- A dictionary is an unordered collection of key:value pairs.
- Keys must be unique and of an immutable type (e.g., strings, numbers, tuples).
- Values can be of any data type.
- Created using curly braces {} or the dict() constructor.

# Common Operations

- Access value: *dict[key]*
- Add/update entry: *dict[key] = value*
- Delete entry: *del dict[key]*
- List keys: *dict.keys()*
- Check key existence: *'key' in dict*

# Looping Techniques

**Different Looping techniques**

- Way 1: Using enumerate()
- Way 2: Using zip()
- Way 3: Using iteritem()
- Way 4: Using items()
- Way 5: Using sorted()
- Way 6: Using reversed()

# More on Conditions

**Advanced Comparison Operators**

- in, not in: Check membership in a sequence.
- is, is not: Check object identity (useful for mutable types like lists).
- Chained comparisons: a < b == c
  - same as a < b and b == c.

**Boolean Operators**

- and, or, not: Combine comparisons.
  - *not* has highest priority, *or* the lowest.
  - Parentheses clarify complex conditions.
- Short-circuit behavior:
  - *and* stops at first *False*, *or* stops at first *True*.

**Usage Tips**

- Comparisons can be stored in variables.
- Assignment not allowed inside expressions.

# Comparing Sequences & Other Types

## Comparing Sequences and Other Types

### Lexicographical Comparison

- Sequences are compared element by element from left to right, and comparison stops when items differ or one sequence ends.
- Recursive comparison if elements are sequences themselves.
- If equal till end, the shorter sequence is smaller.

### Ordering Rules

- Strings use ASCII ordering.
- Comparisons are NOT allowed between different types, but were allowed in python2 and the results were:
  - Deterministic, based on type names.
  - May not be meaningful (e.g., list < string is True).

### Numeric Type Comparison

- Mixed numeric types (int, float) are compared by value.
  - Example: 1 == 1.0 → True

# Modules

**What is a Module?**

- A module is a file containing Python definitions and statements.
- Used to organize and reuse code across scripts and projects.
- Filename ends with .py (e.g., fibo.py).

**Why Use Modules?**

- Avoid rewriting the same functions.
- Keep code organized and manageable.
- Enables modular programming – build and reuse components.

# Creating and Importing Modules

```python
def fibonacci_write(n):
    a, b = 0, 1
    while b < n:
        print(b, end=' ')
        a, b = b, a + b

def fibonacci_return(n):
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a + b
    return result

if __name__ == "__main__":
    fibonacci_write(1000)
```

```python
# Import entire module
import fibonacci as fib
fib.fibonacci_write(1000)
print(fib.fibonacci_return(100))
```

[22] ✓ 0.0s Python

```
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 [1, 1, 2, 3, 5, 8, 13, 21,
```

```python
# Import specific functions
from fibonacci import fibonacci_write, fibonacci_return
fibonacci_write(500)
```

[23] ✓ 0.0s Python

```
1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

```python
# Import all names
from fibonacci import *
```

[24] ✓ 0.0s Python

```python
import fibonacci
print(fibonacci.__name__)  # 'fibonacci'
```

[26] ✓ 0.0s Python

```
fibonacci
```

# Module Behavior

- Executable statements in a module are run once during first import.
- Each module has its own symbol table, used by functions inside it.
- You can access global variables using:
  - *module_name.variable_name*

**Importing Other Modules**

- Modules can also import other modules.
- Conventionally, all import statements go at the top of the file.

# Executing Modules as Scripts

**Running a Module as a Script**

- Modules can act as both reusable code and standalone programs.

**Special Variable: __name__**

- When a module is run directly, "main" is executed :
  - *if __name__ == "__main__":*
    *fibonacci_write(1000)*
- It is not executed if the module is imported.

# THANK YOU