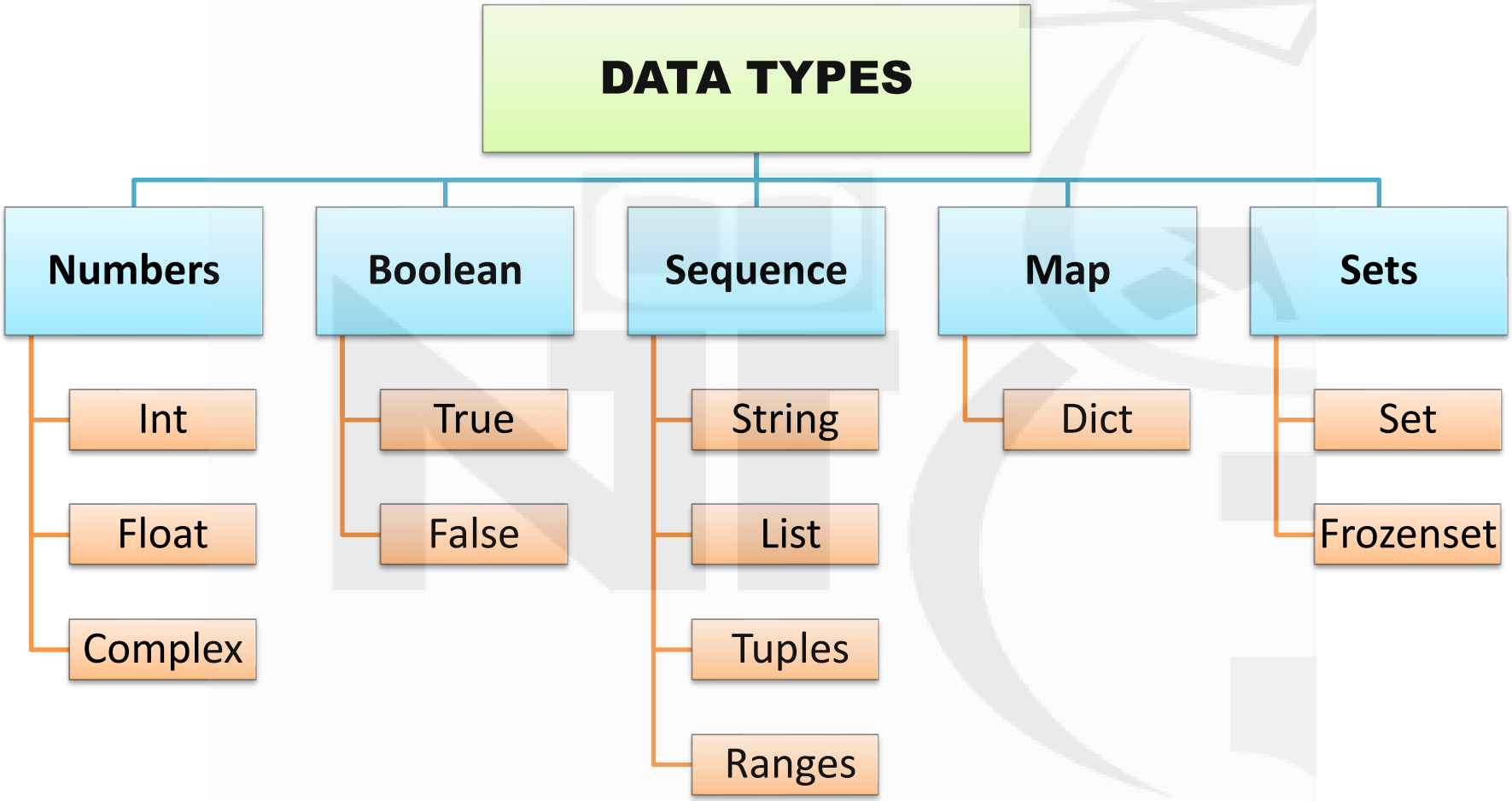
Strings in Python

MUKESH KUMAR


DATA TYPES
Numbers
Boolean
Sequence
Map
Sets
Int
Float
Complex
True
False
String
List
Tuples
Ranges
Dict
Set
Frozenset

Introduction to Strings

- A **string** is a sequence of characters enclosed within quotes.
- Strings can contain any characters, including **letters, numbers, symbols, and spaces**.
- Strings can be defined using : (single, double, triple quotes)

```
str1 = 'Hello'
str2 = "World"
str3 = '''Multiline
String'''
```

String Characterstics

- **Immutable:**
  - Strings in Python are immutable, meaning their values cannot be changed after creation. You can create a new string with the desired modifications.
- **Ordered:**
  - Strings maintain the order of characters. Each character in the string has a specific position, and you can access them using indexing.
- **Iterable:**
  - You can iterate through the characters of a string using loops or comprehension.
- **Concatenation:**
  - Strings can be concatenated using the + operator.
- **String Methods:**
  - Python provides a variety of methods for working with strings, such as len(), lower(), upper(), strip(), split(), and more.

String Examples

- Single quotes:

```
str1 = 'Hello, World!'
print(str1)  # Output: Hello, World!
```

- Double quotes

```
str2 = "Python Programming"
print(str2)  # Output: Python Programming
```

String Examples

- Double quotes is also useful with apostrophe to avoid syntax errors :

```
str3 = "It's a beautiful day!"
print(str3)  # Output: It's a beautiful day!
```

String Examples

- Triple quotes are used to define **multiline strings**

- Triple single quotes:

```
multiline_str1 = '''This is a
multi-line string
in Python.'''
print(multiline_str1)
```

String Example

- Triple double quotes:

```
multiline_str2 = """You can use both 'single' and "double" quotes easily."""
print(multiline_str2)
```

String Indexing & Slicing

- Strings are indexed (0-based and negative indexing)
- Slicing syntax: **string[start:end:step]**

```
text = "Python"
print(text[0])  # P
print(text[-1]) # n
print(text[1:4]) # yth
```

String Methods

Common string methods:

- .upper(), .lower()
- .strip(), .replace()
- .find(), .index()
- .startswith(), .endswith()
- .Split()

String Formatting

- String formatting is a powerful feature in Python
- It allows us to **insert variables** into strings dynamically, making the code more readable and maintainable.

- Python provides multiple ways to format strings.

```
name = "Alice"
age = 25
print(f"My name is {name} and I am {age} years old.")
```

String Formatting

- Using format() Method
- Using f-strings (Formatted String Literals)
- Using % Formatting (Old Method)

USING FORMAT() METHOD

Using format() Method

- The format() method replaces placeholders {} in a string with values provided in the .format() function.

- Example:

```
name = "Alice"
age = 25
formatted_str = "My name is {} and I am {} years old.".format(name, age)
print(formatted_str)
```

- Output:

```
My name is Alice and I am 25 years old.
```

Using Positional Arguments

- You can refer to arguments using **index numbers** inside {}.
- **Example:**

```
print("I love {1} more than {0}.".format("Java", "Python"))
```

  - Here, {1} refers to "Python" and {0} refers to "Java.“

- **Output :**

```
I love Python more than Java.
```

Using Named Arguments

- Instead of index numbers, you can use **keywords**.

- Example:

```
print("My name is {name} and I am {age} years old.".format(name="Bob", age=30))
```

- Output:

```
My name is Bob and I am 30 years old.
```

USING F-STRING

Using F-String (Formatted String Literals)

- **Introduced in Python 3.6**, f-strings are the most **concise and readable** way to format strings.
- They are prefixed with f or F before the string and allow direct **variable interpolation** inside {}.

Using F-String Examples

- Example:

```
name = "Charlie"
age = 28
print(f"My name is {name} and I am {age} years old.")
```

- Output:

```
My name is Charlie and I am 28 years old.
```

Evaluating Expressions in f-strings

- You can **perform calculations** inside {}.

- Example:

```
a = 5
b = 10
print(f"The sum of {a} and {b} is {a + b}.")
```

- Output:

```
The sum of 5 and 10 is 15.
```

Formatting Numbers in f-strings

- You can format numbers just like in format().

- Example:

```
pi = 3.1415926535
print(f"Pi rounded to 2 decimal places: {pi:.2f}")
```

- Output

```
Pi rounded to 2 decimal places: 3.14
```

- **Using f-strings with Lists**
- **Using f-strings with Dictionaries**

USING % FORMATTING (OLD METHOD)

Using % Formatting

- This method is **less preferred** but still works in Python.

- Example:

```
name = "Eve"
age = 22
print("My name is %s and I am %d years old." % (name, age))
```

- Output:

```
My name is Eve and I am 22 years old.
```

Key Takeaways

- **f-strings** (f"...") are the **best choice** for formatting in modern Python (Python 3.6+).
- .format() is **good** but more **verbose** than f-strings.
- % formatting is **old and not recommended**.
- Python provides **powerful formatting options** for text alignment, number formatting, and embedding expressions.

String Operations

- **String concatenation:** "Hello" + " " + "World“
- **Repetition:** "Python " * 3
- **Membership check:** "Py" in "Python"

String Iteration

- Using a for Loop

```
for char in "Python":
    print(char)
```

- Using enumerate() for Index and Character

```
text = "Python"
for index, char in enumerate(text):
    print(f"Index: {index}, Character: {char}")
```

String Iteration

- Using while Loop with Index:

```
text = "Python"
index = 0
while index < len(text):
    print(text[index])
    index += 1
```

Escape Sequences

- Refer to the jupyter notebook

Practice Questions

- Refer the jupyter notebook