

**VIDHI ROHIRA**  
**S.Y B.TECH**  
**COMPUTER**  
**ENGINEERING**

**231071052**  
**BATCH - C**

**PD LAB 9**

# LABORATORY 9

**AIM:-** To study and implement all data types in Python.

## THEORY:-

### Q] WHAT ARE DATA TYPES?

In Python, data types are categories of values that determine the operations that can be performed on them. Here are the main built-in data types:

#### 1. Numeric Types:

- **int**: Integer values (e.g., 5, -3)
- **float**: Floating-point numbers (e.g., 3.14, -0.001)
- **complex**: Complex numbers (e.g., 2 + 3j)

#### 2. Sequence Types:

- **str**: Strings, which are sequences of characters (e.g., "Hello")
- **list**: Ordered, mutable collections (e.g., [1, 2, 3])
- **tuple**: Ordered, immutable collections (e.g., (1, 2, 3))

### 3. Mapping Type:

- **dict**: Unordered collections of key-value pairs (e.g., `{'key': 'value'}`)

### 4. Set Types:

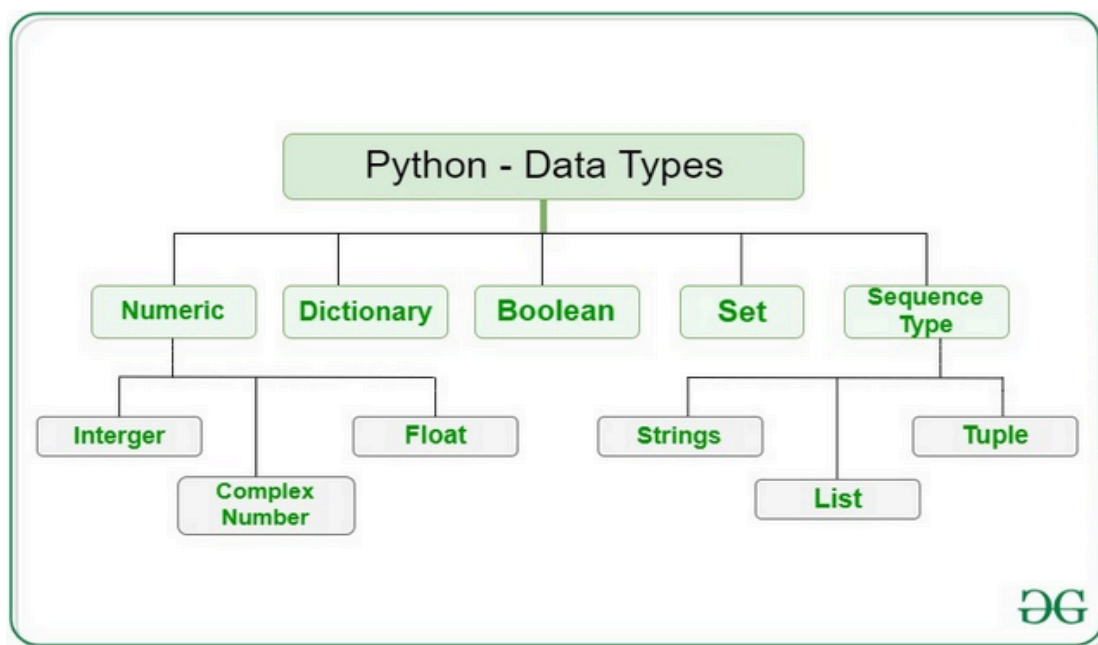
- **set**: Unordered collections of unique elements (e.g., `{1, 2, 3}`)
- **frozenset**: Immutable version of a set

### 5. Boolean Type:

- **bool**: Represents truth values (`True` or `False`)

### 6. None Type:

- **NoneType**: Represents the absence of a value (e.g., `None`)



Primitive Data Types	Non-Primitive Data Types
Immutable (Cannot be changed)	Mutable (Can be changed)
Stored directly in the variable	Stored by reference
number, string, boolean, null, undefined, symbol	object, array
Checked by value	Checked by reference
Fixed size	Dynamic size
Simple operations (copy by value)	Complex operations (copy by reference)
Used for simple data values	Used for more complex structures and collections

## CODE AND EXAMPLES:

### 1] Type Checking:

```
x = 5
print(type(x))
```

#### Output

```
<class 'int'>
```

### 2] Type Conversion:

```
# Convert float to int
int_value = int(3.14)
print("Converted float to int:", int_value) # Output: 3

# Convert int to float
float_value = float(5)
print("Converted int to float:", float_value) # Output: 5.0
```

```
# Convert int to str
str_value = str(100)
print("Converted int to str:", str_value) # Output: '100'
```

Output:

```
⇒ Converted float to int: 3
   Converted int to float: 5.0
   Converted int to str: 100
```

### 3] Custom Class:

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

person = Person("Alice", 30)
print("Custom Class - Person:", person.name, person.age) # Output: Alice
30
```

Output:

```
⇒ Custom Class - Person: Alice 30
```

### 4] Named Tuples:

```
from collections import namedtuple

Point = namedtuple('Point', ['x', 'y'])
p = Point(10, 20)
print("Named Tuple - Point:", p) # Output: Point(x=10, y=20)
```

Output:

```
⇒ Named Tuple - Point: Point(x=10, y=20)
```

## 5]Data Classes:

```
from dataclasses import dataclass

@dataclass
class Product:
    name: str
    price: float

product = Product("Laptop", 999.99)
print("Data Class - Product:", product) # Output: Product(name='Laptop', price=999.99)
```

Output:

```
➦ Data Class - Product: Product(name='Laptop', price=999.99)
```

## 6]Typed Dict:

```
from typing import TypedDict

class Employee(TypedDict):
    name: str
    id: int

employee = Employee(name="Bob", id=123)
print("Typed Dict - Employee:", employee) # Output: {'name': 'Bob', 'id': 123}
```

Output:

```
➦ Typed Dict - Employee: {'name': 'Bob', 'id': 123}
```

## 7]Union Types:

```
from typing import Union

def get_value() -> Union[int, str]:
    return "42" # Could also return an int
```

```
value = get_value()
print("Union Type - Value:", value) # Output: 42
```

Output:

```
➦ Union Type - Value: 42
```

## 8]Optional types:

```
from typing import Optional

def find_item(id: int) -> Optional[str]:
    return None # Simulating that the item is not found

item = find_item(1)
print("Optional Type - Item:", item) # Output: None
```

Output:

```
➦ Optional Type - Item: None
```

## 9]Generators:

```
def generate_numbers(n):
    for i in range(n):
        yield i

gen = generate_numbers(5)
print("Generator values:", list(gen)) # Output: [0, 1, 2, 3, 4]
```

Output:

```
➦ Generator values: [0, 1, 2, 3, 4]
```

## 10] Lambda function:

```
square = lambda x: x ** 2
print("Lambda function - Square of 5:", square(5))  # Output: 25
```

Output:

```
➞ Lambda function - Square of 5: 25
```

## 11] List Comprehensions:

```
squares = [x ** 2 for x in range(10)]
print("List Comprehension - Squares:", squares)  # Output: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Output:

```
➞ List Comprehension - Squares: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

## 12] Sets and frozensets:

```
my_set = {1, 2, 3, 3, 4}
print("Set (unique values):", my_set)  # Output: {1, 2, 3, 4}

my_frozenset = frozenset([1, 2, 3, 3, 4])
print("Frozenset (immutable):", my_frozenset)  # Output: frozenset({1, 2, 3, 4})
```

Output:

```
➞ Set (unique values): {1, 2, 3, 4}
   Frozenset (immutable): frozenset({1, 2, 3, 4})
```

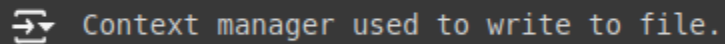
## 13] Contexts Managers:

```
with open('example.txt', 'w') as f:
    f.write("Hello, world!")

print("Context manager used to write to file.")
```



Output:

A terminal window with a dark background. It shows a message: "Context manager used to write to file." preceded by a small icon of a document with a double-headed arrow.

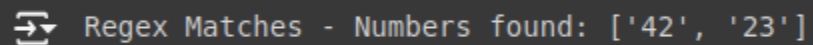
Context manager used to write to file.

14]Regular expression:

```
import re

pattern = r'\d+'
text = "There are 42 apples and 23 oranges."
matches = re.findall(pattern, text)
print("Regex Matches - Numbers found:", matches) # Output: ['42', '23']
```

Output:

A terminal window with a dark background. It shows the output of a Python script: "Regex Matches - Numbers found: ['42', '23']" preceded by a small icon of a document with a double-headed arrow.

Regex Matches - Numbers found: ['42', '23']

## **CONCLUSION : –**

In this lab session, we covered Python data types, focusing on primitive types like integers (int), floats (float), strings (str), booleans (bool), and NoneType, all of which are immutable. We also explored non-primitive types such as lists (list), tuples (tuple), dictionaries (dict), and sets (set), which are mutable and can store multiple values. Additionally, we introduced constructs like generators, lambda functions, list comprehensions, context managers, and regular expressions. This session provided valuable insights into managing and manipulating data in Python.