# Learn Python Programming from Scratch

## *Topic: Data Types in Python*

## 1. What are Data Types?

In Python, **data types** are classifications that specify what type of value a variable can hold and what operations can be performed on it. Python is dynamically typed, meaning you don't need to explicitly declare variable types - Python automatically determines the type based on the value assigned.

Understanding data types is crucial because:

- They determine what operations are valid on variables
- They affect memory usage and performance
- They help prevent errors in your code
- They make your code more readable and maintainable

## 2. Python's Built-in Data Types

Python provides several built-in data types organized into categories:

**Numeric Types:**

- `int` - Integer numbers
- `float` - Decimal numbers
- `complex` - Complex numbers

**Text Type:**

- `str` - Strings (text)

**Boolean Type:**

- `bool` - True/False values

**Sequence Types:**

- `list` - Ordered, mutable collections
- `tuple` - Ordered, immutable collections
- `range` - Sequence of numbers

**Mapping Type:**

- `dict` - Key-value pairs

**Set Types:**

- `set` - Unordered collection of unique elements

- `frozenset` - Immutable set

**Binary Types:**

- `bytes`, `bytearray`, `memoryview`

**None Type:**

- `NoneType` - Represents absence of value

## 3. Numeric Data Types

In [1]:

```python
# Numeric Data Types Examples

# Integer (int) - Whole numbers
integer_num = 10
negative_int = -25
large_int = 1000000

# Float (float) - Decimal numbers
float_num = 10.5
negative_float = -3.14
scientific_notation = 2.5e3  # 2500.0

# Complex (complex) - Numbers with real and imaginary parts
complex_num = 2 + 3j
complex_num2 = complex(4, 5)  # 4+5j

# Check the types
print("=== NUMERIC TYPES ===")
print(f"integer_num = {integer_num}, type = {type(integer_num)}")
print(f"float_num = {float_num}, type = {type(float_num)}")
print(f"complex_num = {complex_num}, type = {type(complex_num)}")

# Mathematical operations work on numeric types
print(f"\nMath operations:")
print(f"10 + 5.5 = {10 + 5.5}")  # int + float = float
print(f"20 / 4 = {20 / 4}")      # Division always returns float
print(f"20 // 3 = {20 // 3}")    # Floor division
print(f"2 ** 3 = {2 ** 3}")      # Exponentiation
```

```
=== NUMERIC TYPES ===
integer_num = 10, type = <class 'int'>
float_num = 10.5, type = <class 'float'>
complex_num = (2+3j), type = <class 'complex'>

Math operations:
10 + 5.5 = 15.5
20 / 4 = 5.0
20 // 3 = 6
2 ** 3 = 8
```

## 4. Text Data Type (String)

Strings are sequences of characters enclosed in quotes. Python supports single, double, and triple quotes for strings.

In [2]:

```python
# String Data Type Examples

# Different ways to create strings
single_quote = 'Hello, Python!'
double_quote = "Hello, World!"
triple_quote = """This is a
multiline string"""

# String with escape characters
escaped_string = "He said, \"Python is awesome!\""
newline_string = "First line\nSecond line"

print("=== STRING TYPE ===")
print(f"single_quote = {single_quote}, type = {type(single_quote)}")
print(f"Length of string: {len(single_quote)}")

# String operations
print(f"\nString operations:")
print(f"Concatenation: {'Hello' + ' ' + 'World'}")
print(f"Repetition: {'Python! ' * 3}")
print(f"Uppercase: {'python'.upper()}")
print(f"Lowercase: {'PYTHON'.lower()}")

# String indexing and slicing
text = "Python Programming"
print(f"\nString indexing:")
print(f"First character: {text[0]}")
print(f"Last character: {text[-1]}")
print(f"Slice [0:6]: {text[0:6]}")
print(f"Slice [7:]: {text[7:]}")

# Check if it's a string
print(f"Is 'hello' a string? {isinstance('hello', str)}")
```

```
=== STRING TYPE ===
single_quote = Hello, Python!, type = <class 'str'>
Length of string: 14

String operations:
Concatenation: Hello World
Repetition: Python! Python! Python!
Uppercase: PYTHON
Lowercase: python

String indexing:
First character: P
Last character: g
Slice [0:6]: Python
Slice [7:]: Programming
Is 'hello' a string? True
```

## 5. Boolean Data Type

Boolean data type represents truth values: `True` or `False`. They are commonly used in conditional statements and logical operations.

In [3]:

```python
# Boolean Data Type Examples
# Boolean values
is_active = True
is_complete = False

print("=== BOOLEAN TYPE ===")
print(f"is_active = {is_active}, type = {type(is_active)}")
print(f"is_complete = {is_complete}, type = {type(is_complete)}")

# Boolean operations
print(f"\nBoolean operations:")
print(f"True and False = {True and False}")
print(f"True or False = {True or False}")
print(f"not True = {not True}")

# Boolean conversion from other types
print(f"\nBoolean conversion:")
print(f"bool(1) = {bool(1)}")        # Non-zero numbers are True
print(f"bool(0) = {bool(0)}")        # Zero is False
print(f"bool('hello') = {bool('hello')}")  # Non-empty strings are True
print(f"bool('') = {bool('')}")      # Empty string is False
print(f"bool([1,2,3]) = {bool([1,2,3])}")  # Non-empty lists are True
print(f"bool([]) = {bool([])}")      # Empty list is False
```

```
=== BOOLEAN TYPE ===
is_active = True, type = <class 'bool'>
is_complete = False, type = <class 'bool'>

Boolean operations:
True and False = False
True or False = True
not True = False

Boolean conversion:
bool(1) = True
bool(0) = False
bool('hello') = True
bool('') = False
bool([1,2,3]) = True
bool([]) = False
```

## 6. Type Checking and Conversion

Python provides built-in functions to check and convert between data types.

In [4]:

```python
# Type Checking and Conversion Examples

# Original values
original_int = 42
original_float = 3.14
original_string = "123"
original_bool = True

print("=== TYPE CHECKING ===")
print(f"type({original_int}) = {type(original_int)}")
print(f"type({original_float}) = {type(original_float)}")
print(f"type('{original_string}') = {type(original_string)}")
print(f"type({original_bool}) = {type(original_bool)}")

# Using isinstance() function
```

```python
print(f"\nUsing isinstance():")
print(f"isinstance(42, int) = {isinstance(42, int)}")
print(f"isinstance(3.14, float) = {isinstance(3.14, float)}")
print(f"isinstance('hello', str) = {isinstance('hello', str)}")

print("\n=== TYPE CONVERSION ===")
# Type conversion (casting)
print(f"int('123') = {int('123')}")          # String to int
print(f"float(42) = {float(42)}")            # Int to float
print(f"str(123) = '{str(123)}'")            # Int to string
print(f"bool(1) = {bool(1)}")                # Int to bool
print(f"int(True) = {int(True)}")            # Bool to int

# Be careful with conversions that might fail
try:
    result = int("hello")  # This will raise ValueError
except ValueError as e:
    print(f"Error converting 'hello' to int: {e}")
```

```
=== TYPE CHECKING ===
type(42) = <class 'int'>
type(3.14) = <class 'float'>
type('123') = <class 'str'>
type(True) = <class 'bool'>

Using isinstance():
isinstance(42, int) = True
isinstance(3.14, float) = True
isinstance('hello', str) = True

=== TYPE CONVERSION ===
int('123') = 123
float(42) = 42.0
str(123) = '123'
bool(1) = True
int(True) = 1
Error converting 'hello' to int: invalid literal for int() with base 10: 'hello'
```

---

## Course Information

**Learn Python Programming from Scratch**

*Author:* Prakash Ukhalkar

*Topic:* Python Fundamentals - Data Types

---

*Built with ❤ for the Python learning community*