PYTHON DATA TYPES

In Python, int, float, complex, bool, and str are the fundamental built-in data types that allow you to represent and manipulate different kinds of data.

1. int (Integer)

- The int data type is used to represent whole numbers, both positive and negative, without any decimal point.

CODE:

```
x = 10          # Positive integer
y = -42         # Negative integer
z = 0           # Zero
```

Problem: You can perform mathematical operations on integers like addition, subtraction, multiplication, division, etc.

```
result = 5 + 10  # 15
```

2. float (Floating-Point)

- The float data type is used to represent real numbers (i.e., numbers with decimals).

CODE:

```
a = 3.14        # Positive float
b = -0.001      # Negative float
c = 2.0         # Float with a value of 2
```

Problem: Like integers, you can perform arithmetic operations, but the result may have a decimal point.

```
result = 3.14 * 2  # 6.28
```

3. complex (Complex Number)

- The complex data type represents complex numbers, which consist of a real part and an imaginary part. Complex numbers are written as a + bj, where a is the real part and b is the imaginary part.

CODE:

```
x = 1 + 2j      # Complex number with real part 1 and imaginary part 2
y = -3 - 4j     # Complex number with real part -3 and imaginary part -4
```

Problem: You can perform arithmetic operations on complex numbers, and Python will handle the real and imaginary parts accordingly.

```
result = (1 + 2j) + (3 + 4j)  # (4 + 6j)
```

4. bool (Boolean)

- The bool data type is used to represent one of two truth values: True or False. It is primarily used in logical operations and conditional statements.

CODE:

```
x = True        # Boolean True
y = False       # Boolean False
```

Problem: Booleans are often used in logical comparisons.

```
result = (5 > 3)  # True
result = (2 == 3) # False
```

5. str (String)

- The str data type is used to represent a sequence of characters (text). Strings can be enclosed in single quotes (') or double quotes (").

CODE:

```
name = "Alice"   # String with double quotes
message = 'Hello, World!'  # String with single quotes
```

Problem: Strings support a variety of operations, such as concatenation, slicing, and methods to manipulate text.

```
greeting = "Hello, " + "Alice"  # "Hello, Alice"
length = len("Python")          # 6
```

Summary of Data Types:

Data Type	Example	Description
int	`10`, `-42`, `0`	Represents whole numbers without decimals.
float	`3.14`, `-0.001`	Represents numbers with decimal points.
complex	`1 + 2j`, `-3 - 4j`	Represents complex numbers with a real and imaginary part.
bool	`True`, `False`	Represents boolean values, used in logical operations.
str	`"Hello"`, `'Alice'`	Represents sequences of characters (text).

TYPE CASTING

- In Python, **type casting** refers to converting one data type to another.
- This is particularly useful when you need to work with different data types in your program, for example, converting a string to an integer or a float to an integer.
- Python provides both **implicit** and **explicit** type casting:

1. Implicit Type Casting (Automatic Conversion)

Python automatically converts data types when necessary (this happens when a lower data type is used in a context where a higher data type is expected).

Code:

```
x = 5     # int
y = 2.5   # float
z = x + y # Implicit conversion of int to float
print(z)  # Output: 7.5 (float)
```

In this example, Python implicitly converts x (an int) to a float before performing the addition with y, resulting in a float.

2. Explicit Type Casting (Manual Conversion)

- Explicit type casting is when you manually convert a value from one type to another using **casting functions**.

You can do these using functions like:

int() to convert to an integer
float() to convert to a float
str() to convert to a string
bool() to convert to a boolean

CODE:

Converting to Integer (int()):

```
x = "10"        # string
y = int(x)      # explicitly cast to integer
print(y)        # Output: 10 (int)
```

Converting to Float (float()):

```
x = "3.14"      # string
y = float(x)    # explicitly cast to float
print(y)        # Output: 3.14 (float)
```

Converting to String (str()):

```
x = 100         # integer
y = str(x)      # explicitly cast to string
print(y)        # Output: "100" (string)
```

Converting to Boolean (bool()):

```
x = 1           # non-zero number, converts to True
y = bool(x)
print(y)        # Output: True

x = 0           # zero converts to False
y = bool(x)
print(y)        # Output: False
```

3. Common Type Conversion Scenarios

String to Integer or Float:

If you have a string that represents a number, you can convert it to an integer or float.

```
string_value = "123"
int_value = int(string_value)     # 123 (int)
float_value = float(string_value) # 123.0 (float)
```

Integer/Float to String:

When you want to convert numbers back to a string for display or concatenation.

```
num = 45
str_value = str(num)  # "45" (string)
```

String to Boolean:

Non-empty strings are converted to True, while an empty string is converted to False.

```
x = "hello"
y = bool(x)   # True (non-empty string)
z = bool("")  # False (empty string)
```

Boolean to Integer/Float:

In Python, True is equivalent to 1 and False is equivalent to 0 when converted to numbers.

```
bool_val = True
int_val = int(bool_val)   # 1
float_val = float(bool_val)  # 1.0
```

4. Error Handling in Type Casting

Type casting can raise errors if the value cannot be converted to the target type. For example, trying to convert a non-numeric string to an integer will result in a ValueError.

Code:

```
x = "hello"
try:
    y = int(x)  # This will raise a ValueError
except ValueError as e:
    print(f"Error: {e}")  # Output: Error: invalid literal for int() with base 10: 'hello'
```

Function	Description	Example
`int()`	Converts a value to an integer	`int("12.34")` → `12`
`float()`	Converts a value to a float	`float("3")` → `3.0`
`str()`	Converts a value to a string	`str(100)` → `"100"`
`bool()`	Converts a value to a boolean	`bool(0)` → `False`

Key Points:

Implicit casting happens automatically when Python upgrades data types (e.g., from int to float).
Explicit casting requires using casting functions like int(), float(), str(), etc.
You should handle potential errors when casting, especially when converting strings to numbers.