Type Conversion in Python

Premanand S

Assistant Professor
School of Electronics Engineering
Vellore Institute of Technology
Chennai Campus

premanand.s@vit.ac.in

November 12, 2024

Introduction to Type Conversion

- **Type Conversion** is the process of converting a value from one data type to another.
- Python provides **implicit** (automatic) and **explicit** (manual) type conversions.
- Useful for handling user input, mathematical operations, and working with mixed data types.

Types of Type Conversion

- **Implicit Type Conversion**: Automatically handled by Python, requiring no explicit instructions.
- **Explicit Type Conversion**: Performed by the programmer using specific functions like `int()`, `float()`, etc.

Implicit Type Conversion

- Python automatically converts types to prevent data loss where possible.
- Common with numeric operations and string concatenation.

Example (Python Code)

```
num_int = 10
num_float = 2.5
result = num_int + num_float  # Converts to float
print(result)  # Output: 12.5
```

Explicit Type Conversion

- Requires specific functions to convert types.
- Common functions include `int()`, `float()`, `str()`, `bool()`.

Example (Python Code)

```
num_str = "100"
num_int = int(num_str)  # Converts string to integer
print(type(num_int))  # Output: <class 'int'>
```

Common Type Conversion Scenarios

- **User Input**: Converting input from strings to numeric types.
- **Mathematical Operations**: Converting to avoid type errors.
- **String Formatting**: Converting numbers to strings for display.

Example (Python Code)

```
 # Converts input to integer
age = int(input("Enter your age: "))

# Converts integer to string for concatenation
text = "Age: " + str(age)
```

Handling Errors in Type Conversion

- **ValueError** occurs when conversion fails.
- Use `try-except` blocks to handle conversion errors gracefully.

Example (Python Code)

```
try:
    num = int("abc")
except ValueError:
    print("Cannot convert to integer.")
```

Advanced Type Conversion Techniques

- `eval()`: Converts string expressions to actual Python objects. Caution: can be unsafe.
- `ord()` and `chr()`: Convert between characters and ASCII codes.

Example (Python Code)

```
char = 'A'
ascii_code = ord(char)  # 65
new_char = chr(65)  # 'A'
```

Type Conversion in Collections

- Converting lists, tuples, sets, or dictionaries.
- Use `list()`, `tuple()`, `set()`, `dict()` for conversion between collections.

Example (Python Code)

```
str_data = "abc"
list_data = list(str_data)  # ['a', 'b', 'c']
```

Type Conversion in Lists

- Using list comprehensions to convert types in lists.

Example (Python Code)

```
str_list = ["1", "2", "3"]
int_list = [int(x) for x in str_list]  # [1, 2, 3]
```

Type Hints and Conversion

- Type hints specify the expected type for variables or function parameters.
- Use `__annotations__` to check expected types.

Example (Python Code)

```
def add_numbers(a: int, b: int) -> int:
    return a + b
print(add_numbers.__annotations__)
```

Tips and Best Practices for Type Conversion

- **Verify Feasibility**: Check if data can be converted without errors.
- **Handle Exceptions**: Always use try-except blocks where conversions might fail.
- **Avoid Unnecessary Conversions**: Only convert types when necessary.

Improper Type Conversion Examples

- Demonstrate cases where conversion leads to errors or data loss.
- Example: Converting a float to an integer loses decimal information.

Example (Python Code)

```
num_float = 5.67
num_int = int(num_float)  # 5 (loses .67)
```

Automatic Promotion of Types in Expressions

- Python promotes types automatically in expressions involving different types.
- Numeric operations between `int` and `float` promote to `float`.
- Useful when working with custom numeric types like `Decimal` or `Fraction`.

Example (Python Code)

```
from fractions import Fraction
result = Fraction(1, 3) + 0.5  # Promotes to float
```

Complex Numbers and Type Conversion

- Python supports `complex` numbers with real and imaginary parts.
- Converting `int` or `float` to `complex` is allowed.
- Conversion from `complex` to other types is not allowed if the imaginary part is non-zero.

Example (Python Code)

```
c = complex(4, 3)
real_only = int(c.real)  # Converts only real part
```

Handling Inexact Conversions with Decimal and Fraction

- `Decimal` avoids precision issues in floats.
- `Fraction` represents exact rational numbers.

Example (Python Code)

```
from decimal import Decimal
d = Decimal("0.1") + Decimal("0.2")  # Exact 0.3

from fractions import Fraction
f = Fraction(1, 3) + Fraction(1, 6)  # Exact 1/2
```

astype Method in NumPy Arrays

- NumPy's `astype()` method efficiently converts arrays to a new type.

Example (Python Code)

```
import numpy as np
arr = np.array([1.5, 2.3, 3.7])
int_arr = arr.astype(int)  # All elements as int
```

Custom Type Conversion with `__int__`, `__float__`, and `__str__`

- Define `__int__`, `__float__`, and `__str__` in custom classes for flexible conversion.

Example (Python Code)

```
class Currency:
    def __init__(self, amount):
        self.amount = amount

    def __int__(self):
        return int(self.amount)
    def __float__(self):
        return float(self.amount)

c = Currency(99.99)
print(int(c))  # 99
print(float(c))  # 99.99
```

Conclusion

- Type conversion is essential for data handling in Python.
- Understand the differences between implicit and explicit conversion.
- Handle conversions with care to avoid errors and data loss.