Learn Python Programming from Scratch

Topic: Type Conversion in Python

1. What is Type Conversion?

Type conversion (also called **type casting**) is the process of converting a value from one data type to another. Python provides built-in functions to convert between different data types like integers, floats, strings, and booleans.

Type conversion is essential for:

- Processing user input (always strings from `input()`)
- Performing calculations with mixed data types
- Formatting output for display
- Working with data from files or databases
- Ensuring compatibility between different parts of your program

2. Types of Conversion

Implicit Conversion (Automatic):

- Python automatically converts types when needed
- Example: `5 + 3.0` → Python converts `5` to `5.0`

Explicit Conversion (Manual):

- You manually convert using built-in functions
- Example: `int("123")` → Convert string to integer

3. Built-in Conversion Functions

- `int()` - Convert to integer
- `float()` - Convert to floating-point number
- `str()` - Convert to string
- `bool()` - Convert to boolean
- `list()` - Convert to list
- `tuple()` - Convert to tuple
- `set()` - Convert to set

In [4]:
```
# Type Conversion Examples

print("=== EXPLICIT TYPE CONVERSION ===")

# String to Number Conversion
string_number = "123"
string_float = "45.67"
string_negative = "-89"
```

```
print(f"Original strings: '{string_number}', '{string_float}', '{string_negative}'
print(f"int('{string_number}') = {int(string_number)} (type: {type(int(string_numb
print(f"float('{string_float}') = {float(string_float)} (type: {type(float(string_
print(f"int('{string_negative}') = {int(string_negative)} (type: {type(int(string_

# Number to String Conversion
num_int = 42
num_float = 3.14159
print(f"\nNumber to String:")
print(f"str({num_int}) = '{str(num_int)}' (type: {type(str(num_int))})")
print(f"str({num_float}) = '{str(num_float)}' (type: {type(str(num_float))})")

print("\n=== BOOLEAN CONVERSIONS ===")

# Convert to Boolean
test_values = [0, 1, -1, 0.0, 3.14, "", "hello", [], [1,2,3], None]
print("Value           bool()    Type")
print("-" * 35)
for val in test_values:
    bool_val = bool(val)
    print(f"{repr(val):14} {bool_val:8} {type(val).__name__}")

# Convert from Boolean
print(f"\nBoolean to Numbers:")
print(f"int(True) = {int(True)}")
print(f"int(False) = {int(False)}")
print(f"float(True) = {float(True)}")
print(f"str(True) = '{str(True)}'")

print("\n=== IMPLICIT TYPE CONVERSION ===")

# Python automatically converts types in mixed operations
int_num = 10
float_num = 3.5

result = int_num + float_num  # int is automatically converted to float
print(f"Implicit conversion: {int_num} + {float_num} = {result} (type: {type(resul

# Division always returns float
division_result = 10 / 2
print(f"Division result: 10 / 2 = {division_result} (type: {type(division_result)}

print("\n=== PRACTICAL EXAMPLES ===")

# User input processing
print("=== User Input Processing ===")
# Note: In notebooks, we'll simulate user input
user_age = "25"  # Simulating input("Enter your age: ")
user_height = "5.9"  # Simulating input("Enter your height: ")

print(f"Raw input - Age: '{user_age}' (type: {type(user_age)})")
print(f"Raw input - Height: '{user_height}' (type: {type(user_height)})")

# Convert for calculations
age = int(user_age)
height = float(user_height)

next_year_age = age + 1
height_in_cm = height * 30.48  # Convert feet to cm
```

```python
print(f"Processed - Age: {age}, Next year: {next_year_age}")
print(f"Processed - Height: {height} feet = {height_in_cm:.1f} cm")

print("\n=== ERROR HANDLING IN CONVERSION ===")

# Safe conversion with error handling
def safe_int_conversion(value):
    """Safely convert a value to integer with error handling."""
    try:
        return int(value), True
    except (ValueError, TypeError) as e:
        return None, False

# Test safe conversion
test_inputs = ["123", "45.67", "hello", "12.34", None, [1,2,3]]
print("Value        Converted    Success")
print("-" * 35)
for val in test_inputs:
    converted, success = safe_int_conversion(val)
    converted_str = str(converted) if converted is not None else "None"
    print(f"{repr(val):12} {converted_str:12} {success}")

print("\n=== COLLECTION CONVERSIONS ===")

# Convert between collection types
numbers_list = [1, 2, 3, 4, 5]
numbers_string = "12345"
words = "hello world python"

print(f"Original list: {numbers_list}")
print(f"list → tuple: {tuple(numbers_list)}")
print(f"list → set: {set(numbers_list)}")

print(f"\nString: '{numbers_string}'")
print(f"string → list: {list(numbers_string)}")

print(f"\nWords: '{words}'")
print(f"split → list: {words.split()}")
print(f"join list: {' '.join(['Python', 'is', 'awesome'])}")

# Round-trip conversion example
original = [1, 2, 3, 4, 5]
as_string = str(original)
print(f"\nRound-trip example:")
print(f"Original: {original}")
print(f"As string: '{as_string}'")
# Note: eval() is dangerous - don't use in real code!
# back_to_list = eval(as_string)  # DON'T DO THIS!
print("(Converting string back to list requires proper parsing)")

print("\n=== TYPE CHECKING BEFORE CONVERSION ===")

def smart_convert(value, target_type):
    """Convert value to target type with validation."""
    print(f"Converting {repr(value)} to {target_type.__name__}:")

    if isinstance(value, target_type):
        print(f"  Already {target_type.__name__}: {value}")
        return value
```

```
    try:
        result = target_type(value)
        print(f"  Success: {result}")
        return result
    except (ValueError, TypeError) as e:
        print(f"  Error: {e}")
        return None

# Test smart conversion
test_cases = [
    ("123", int),
    (123, int),
    ("45.67", float),
    (45.67, str),
    ([1,2,3], str),
    ("hello", int)
]

for value, target in test_cases:
    smart_convert(value, target)
    print()
```

```
=== EXPLICIT TYPE CONVERSION ===
Original strings: '123', '45.67', '-89'
int('123') = 123 (type: <class 'int'>)
float('45.67') = 45.67 (type: <class 'float'>)
int('-89') = -89 (type: <class 'int'>)

Number to String:
str(42) = '42' (type: <class 'str'>)
str(3.14159) = '3.14159' (type: <class 'str'>)

=== BOOLEAN CONVERSIONS ===
Value           bool()    Type
-----------------------------------
0                     0 int
1                     1 int
-1                    1 int
0.0                   0 float
3.14                  1 float
''                    0 str
'hello'               1 str
[]                    0 list
[1, 2, 3]             1 list
None                  0 NoneType

Boolean to Numbers:
int(True) = 1
int(False) = 0
float(True) = 1.0
str(True) = 'True'

=== IMPLICIT TYPE CONVERSION ===
Implicit conversion: 10 + 3.5 = 13.5 (type: <class 'float'>)
Division result: 10 / 2 = 5.0 (type: <class 'float'>)

=== PRACTICAL EXAMPLES ===
=== User Input Processing ===
Raw input - Age: '25' (type: <class 'str'>)
Raw input - Height: '5.9' (type: <class 'str'>)
Processed - Age: 25, Next year: 26
Processed - Height: 5.9 feet = 179.8 cm

=== ERROR HANDLING IN CONVERSION ===
Value        Converted    Success
-----------------------------------
'123'        123          True
'45.67'      None         False
'hello'      None         False
'12.34'      None         False
None         None         False
[1, 2, 3]    None         False

=== COLLECTION CONVERSIONS ===
Original list: [1, 2, 3, 4, 5]
list → tuple: (1, 2, 3, 4, 5)
list → set: {1, 2, 3, 4, 5}

String: '12345'
string → list: ['1', '2', '3', '4', '5']

Words: 'hello world python'
```

```
split → list: ['hello', 'world', 'python']
join list: Python is awesome

Round-trip example:
Original: [1, 2, 3, 4, 5]
As string: '[1, 2, 3, 4, 5]'
(Converting string back to list requires proper parsing)

=== TYPE CHECKING BEFORE CONVERSION ===
Converting '123' to int:
  Success: 123

Converting 123 to int:
  Already int: 123

Converting '45.67' to float:
  Success: 45.67

Converting 45.67 to str:
  Success: 45.67

Converting [1, 2, 3] to str:
  Success: [1, 2, 3]

Converting 'hello' to int:
  Error: invalid literal for int() with base 10: 'hello'
```

Key Takeaways

- **Explicit conversion** uses functions like `int()`, `float()`, `str()`, `bool()`
- **Implicit conversion** happens automatically in mixed operations
- `input()` **always returns strings** - convert to numbers for calculations
- **Error handling** is crucial - conversions can fail with `ValueError`
- **Boolean conversion**: `0`, `""`, `[]`, `None` are falsy; everything else is truthy
- **Division always returns float** even with whole number results
- **Type checking** with `isinstance()` before conversion prevents errors

Conversion Reference Table

From → To	Function	Example	Result	Notes
str → int	`int()`	`int("123")`	`123`	Must be valid integer
str → float	`float()`	`float("3.14")`	`3.14`	Must be valid number
int → str	`str()`	`str(123)`	`"123"`	Always works
int → float	`float()`	`float(123)`	`123.0`	Always works
float → int	`int()`	`int(3.14)`	`3`	Truncates decimal
any → bool	`bool()`	`bool(123)`	`True`	0/empty = False

Common Conversion Pitfalls

1. **ValueError**: `int("hello")` - Invalid string format
2. **Data loss**: `int(3.9)` → `3` (decimal part lost)
3. **Type assumptions**: Always check input before conversion
4. **Float precision**: Some decimals can't be represented exactly
5. **Empty collections**: `int("")` fails, but `bool("")` returns `False`

Practice Exercises

1. **Input Validator**: Create a function that safely converts user input to numbers
2. **Data Type Analyzer**: Build a program that identifies and converts data types
3. **Calculator Enhancement**: Handle string inputs and convert appropriately
4. **File Data Processor**: Convert string data from files to appropriate types
5. **Form Validator**: Validate and convert web form data

Course Information

Learn Python Programming from Scratch
Author: Prakash Ukhalkar
Topic: Python Fundamentals - Type Conversion

Built with ❤️ for the Python learning community