AIMERZ.ai
Aim.Act.Achieve

NOTES

Type Conversion and Conditions

Type Conversion and Conditions

Python Type Conversion: Implicit and Explicit Type Conversion

In Python, type conversion is the process of converting one data type into another. This is important because sometimes operations require specific data types, and understanding how to convert types ensures that your code works as expected.

We'll break this down into two main concepts:

- Implicit type conversion (automatic)
- Explicit type conversion (casting)

1. Implicit Type Conversion (Automatic Conversion)

Implicit type conversion occurs when Python automatically converts one data type to another during an operation. This usually happens when combining different types in expressions, and Python converts them without the need for explicit instructions from the programmer. It's also called **coercion**.

Python does this to avoid data loss and ensure accuracy during operations.

Example:

```python
x = 10   # Integer
y = 3.5  # Float

# Python automatically converts 'x' to a float to match 'y'
result = x + y

print(result)  # Output: 13.5 (float)
print(type(result))  # Output: <class 'float'>
```

In the above example, x (an integer) is automatically converted to a float during the addition operation because y is a float.

Notes:

- Python automatically converts smaller data types to larger ones (e.g., int to float).
- Implicit conversion helps prevent errors when dealing with different types but may lead to unexpected results if not carefully considered.

2. Explicit Type Conversion (Casting)

Explicit type conversion occurs when you manually convert one data type into another using Python's built-in functions. This is useful when you need to control the conversion process and ensure that a specific type is used.

Common functions used for explicit type conversion (casting):

- `int()` : Converts to an integer
- `float()` : Converts to a float
- `str()` : Converts to a string
- `list()` : Converts to a list
- `tuple()` : Converts to a tuple
- `bool()` : Converts to a Boolean

Example of Casting to an Integer:

```python
num = "42"  # String

# Converting string to integer
int_num = int(num)

print(int_num)  # Output: 42
print(type(int_num))  # Output: <class 'int'>
```

In this example, the string "42" is explicitly converted to an integer using int().

Example of Casting a Float to an Integer:

```python
float_num = 7.8  # Float

# Converting float to integer
int_num = int(float_num)

print(int_num)  # Output: 7 (the decimal part is truncated)
print(type(int_num))  # Output: <class 'int'>
```

Here, when converting a float to an integer, the decimal part is removed. This is something candidates should be aware of, as it can lead to data loss.

3. Practical Use Cases of Type Conversion in Python

Converting User Input:

When taking input from a user in Python using input(), the data is always treated as a string. If you need to perform calculations, you must convert it to the appropriate type.

Example:

```python
age = input("Enter your age: ")  # This returns a string
age = int(age)  # Explicitly converting to an integer

print(f"You will be {age + 1} next year!")
```

Without the conversion, trying to add a number to the input would raise a TypeError.

Combining Data Types in Calculations:

When working with data from different sources (e.g., from user input or files), type conversion ensures that operations can proceed smoothly. If not handled properly, mixing data types can lead to errors.

Example:

```python
price = "100.50"  # String
quantity = 3      # Integer

# We need to convert 'price' to a float for multiplication
total_cost = float(price) * quantity

print(total_cost)  # Output: 301.5
```

In this example, if we hadn't converted the string `"100.50"` to a float, multiplying it with the integer `quantity` would have caused an error.

Conditions in Python (If, Else, If-Elif-Else)

In Python, conditional statements allow the program to make decisions based on different conditions. This is crucial in controlling the flow of a program, as it determines which blocks of code get executed depending on whether a condition is true or false. This section explains the **if**, **else**, and **if-elif-else** statements, breaking them down into simple, digestible parts for both technical and non-technical learners.

1. If Statement

The **if** statement allows you to test a condition and execute a block of code if the condition evaluates to `True`. If the condition is `False`, the block is skipped.

Syntax:

```python
if condition:
    # Code block to execute if condition is true
```

Example:

```python
age = 18

if age >= 18:
    print("You are eligible to vote!")
```

In this example, Python checks if the `age` is greater than or equal to 18. If this condition is true, the message "You are eligible to vote!" will be printed.

2. Else Statement

The **else** statement provides an alternative block of code that runs if the condition in the **if** statement is `False`. This helps to define what happens when the condition isn't met.

Syntax:

```python
if condition:
    # Code block to execute if condition is true
else:
    # Code block to execute if condition is false
```

Example:

```python
age = 16

if age >= 18:
    print("You are eligible to vote!")
else:
    print("You are not eligible to vote.")
```

In this case, if the age is less than 18, the program will print "You are not eligible to vote." The **else** statement ensures that some code will always be executed, regardless of the condition's truth value.

3. If-Elif-Else Statement

The **if-elif-else** statement allows for checking multiple conditions. The program checks the first condition with **if**. If it's false, it checks the next condition using **elif** (which stands for "else if"). If none of the conditions are true, the **else** block executes.

Syntax:

```python
if condition1:
    # Code block if condition1 is true
elif condition2:
    # Code block if condition2 is true
else:
    # Code block if none of the conditions are true
```

Example:

```python
age = 20

if age < 12:
    print("You are a child.")
elif age < 18:
    print("You are a teenager.")
else:
    print("You are an adult.")
```

In this example, the program checks if the age is less than 12. If not, it checks if the age is less than 18. If neither condition is true, it defaults to the **else** block, which prints "You are an adult."

4. Nested If Statements

You can also nest **if** statements within other **if** statements to test multiple conditions. This is useful for more complex decision-making.

Example:

```python
age = 20
citizen = True

if age >= 18:
    if citizen:
        print("You are eligible to vote.")
    else:
        print("You are not a citizen, so you cannot vote.")
else:
    print("You are not old enough to vote.")
```

Output:

```
You are eligible to vote
```

Here, the program first checks if the age is greater than or equal to 18. If true, it checks whether the person is a citizen. This allows for more granular control over decision-making.

5. Comparisons in Conditional Statements

It's crucial to understand the various comparison operators used in conditions, as these are the backbone of decision-making:

- **Equal to**: `==`
- **Not equal to**: `!=`
- **Greater than**: `>`
- **Less than**: `<`
- **Greater than or equal to**: `>=`
- **Less than or equal to**: `<=`

Example of Comparison Operators:

```python
x = 10
y = 20

# Equal to (==)
if x == y:
    print("x is equal to y")
else:
    print("x is not equal to y")  # Output: x is not equal to y

# Not equal to (!=)
if x != y:
    print("x is not equal to y")  # Output: x is not equal to y

# Greater than (>)
if y > x:
    print("y is greater than x")  # Output: y is greater than x

# Less than (<)
if x < y:
    print("x is less than y")  # Output: x is less than y

# Greater than or equal to (>=)
if y >= 15:
    print("y is greater than or equal to 15")  # Output: y is greater
than or equal to 15
```

```
# Less than or equal to (<=)
if x <= 10:
    print("x is less than or equal to 10")  # Output: x is less than
or equal to 10
```

6. Logical Operators in Conditions

In Python, logical operators such as **and**, **or**, and **not** can be used to combine multiple conditions.

- **and**: Both conditions must be true.
- **or**: At least one condition must be true.
- **not**: Reverses the result of a condition.

Example with Logical Operators:

Python

```
age = 25
citizen = True
has_voter_id = False

# Logical AND: Both conditions must be true
if age >= 18 and citizen:
    print("You are eligible to vote.")  # Output: You are eligible to
vote.

# Logical OR: At least one condition must be true
if age < 18 or not citizen:
    print("You are not eligible to vote.")
else:
    print("You are eligible to vote.")  # Output: You are eligible to
vote.

# Logical NOT: Reverses the result of the condition
if not has_voter_id:
    print("You need a voter ID to vote.")  # Output: You need a voter
ID to vote.
```

Problem to solve:

```python
score = 85

if score >= 90:
    print("Grade: A")
elif score >= 80:
    print("Grade: B")
elif score >= 70:
    print("Grade: C")
else:
    print("Grade: F")
```

Find the output???

THANK YOU

AIMERZ.ai
Aim.Act.Achieve