Python Type Conversion and Casting

Transform Data Into the Type Your Program Needs

---

1. Why Type Conversion Matters

A Python program constantly receives and produces different kinds of data.

Consider a simple shopping application:

```
product = "Python Course"
price = "49.99"
quantity = "2"
```

At first glance, `price` and `quantity` look like numbers.

But Python sees:

```
price     → str
quantity  → str
```

If you want to calculate the total price, you need numerical values:

```
price = 49.99
quantity = 2

total = price * quantity
```

This is where **type conversion** becomes important.

Type conversion means changing a value from one data type to another.

You will use it when working with:

- user input
- APIs
- files
- databases
- JSON
- forms
- calculations
- configuration values
- external data
- validation

A developer who understands type conversion can move data safely between different parts of an application.

---

2. What Is Type Conversion?

Type conversion is the process of changing a value from one data type to another.

For example:

```
age = "25"

age = int(age)
```

The original value is a string:

```
"25" → str
```

After conversion:

```
25 → int
```

Python provides built-in functions that perform many common conversions.

The most important ones are:

```
int()
float()
str()
bool()
```

There are also constructors for collection types such as:

```
list()
tuple()
set()
dict()
```

Those will become especially useful when you learn Python data structures.

---

3. Conversion vs Casting

The terms **type conversion** and **type casting** are often used interchangeably by beginners and even in many tutorials.

In practical Python programming, you will commonly see statements such as:

```
int("25")
float("10.5")
str(100)
```

described as **casting** or **conversion**.

A useful distinction is:

- **Type conversion** can describe changing data from one type to another generally.
- **Type casting** commonly refers to explicitly converting a value using a type constructor such as `int()`, `float()`, or `str()`.

For Python beginners, the important skill is understanding how explicit conversion works and what happens to the original value.

---

4. Explicit Type Conversion

Python allows you to explicitly request a different type.

Example:

```
age = "25"

age = int(age)
```

You are explicitly telling Python:

> Convert this value into an integer.

Another example:

```
price = "49.99"

price = float(price)
```

Now:

```
"49.99" → 49.99
str      → float
```

---

5. Why Python Does Not Convert Everything Automatically

Suppose you write:

```
age = "25"
print(age + 5)
```

Python does not automatically decide that `"25"` should become `25`.

Why?

Because `"25"` is text.

It could be:

- a number
- an ID
- part of a username
- a product code
- a postal code

For example:

```
product_code = "25"
```

You probably do not want Python to perform mathematical operations on it.

Therefore, Python generally requires you to explicitly state your intention.

---

6. The `int()` Function

`int()` converts compatible values into integers.

Basic example:

```
age = int("25")

print(age)
print(type(age))
```

Output:

```
25
<class 'int'>
```

6.1 String to Integer

This is one of the most common conversions.

```
number = "100"

number = int(number)

print(number + 50)
```

Output:

```
150
```

Without conversion:

```
number = "100"

print(number + 50)
```

Python raises a `TypeError`.

---

7. Converting Floats to Integers

You can also convert a float to an integer.

```
price = 49.99

price = int(price)

print(price)
```

Output:

```
49
```

The decimal portion is discarded.

It does **not** round the number to the nearest integer.

For example:

```
print(int(9.9))
print(int(9.1))
print(int(-9.9))
```

The conversion removes the fractional part toward zero.

Conceptually:

```
9.9   → 9
9.1   → 9
-9.9  → -9
```

If you need mathematical rounding, use `round()` instead.

```
print(round(9.9))
```

Output:

```
10
```

---

8. Converting Booleans to Integers

Python treats boolean values numerically in certain contexts.

```
print(int(True))
print(int(False))
```

Output:

```
1
0
```

Conceptually:

```
True  → 1
False → 0
```

This behavior can be useful in certain calculations, but do not use it simply because it is possible.

Choose the representation that best communicates your program's meaning.

---

9. The `float()` Function

`float()` converts compatible values into floating-point numbers.

Example:

```
price = float("49.99")

print(price)
print(type(price))
```

Output:

```
49.99
<class 'float'>
```

9.1 Integer to Float

```
number = 25

number = float(number)

print(number)
```

Output:

```
25.0
```

The numerical value remains equivalent, but its type changes.

9.2 String to Float

```
temperature = "36.5"

temperature = float(temperature)

print(temperature)
```

Output:

```
36.5
```

This is particularly common when reading values from files, forms, or APIs.

---

10. The `str()` Function

`str()` converts a value into a string.

Example:

```
age = 25

message = "Age: " + str(age)

print(message)
```

Output:

```
Age: 25
```

Without conversion:

```
age = 25

message = "Age: " + age
```

Python raises a `TypeError` because string concatenation expects compatible string values.

---

11. Converting Floats to Strings

```
price = 49.99
```

```
price_text = str(price)

print(price_text)
print(type(price_text))
```

Output:

```
49.99
<class 'str'>
```

The value is now text.

This is useful when you need to:

- display information
- concatenate text
- build messages
- serialize information
- prepare certain values for external systems

---

12. The `bool()` Function

`bool()` converts a value into a boolean.

Example:

```
print(bool(1))
print(bool(0))
```

Output:

```
True
False
```

But boolean conversion has rules that are important to understand.

---

13. Truthy and Falsy Values

Python considers some values **truthy** and others **falsy** when converted to `bool`.

Common falsy values include:

```
False
None
0
0.0
""
```

For example:

```
print(bool(False))
print(bool(None))
print(bool(0))
print(bool(0.0))
print(bool(""))
```

All produce:

```
False
```

Many other values are truthy.

For example:

```
print(bool(1))
print(bool(-10))
print(bool("Python"))
```

Output:

```
True
True
True
```

This becomes extremely useful with `if` statements.

---

14. Boolean Conversion in Real Programs

Suppose a user enters a username.

```
username = input("Enter username: ")

if bool(username):
    print("Username provided")
```

You could also simply write:

```
if username:
    print("Username provided")
```

Python automatically uses the value's truthiness in the condition.

Understanding `bool()` helps you understand why these conditions work.

---

15. Important Difference: `bool("False")`

A common beginner mistake is assuming:

```
bool("False")
```

will produce:

```
False
```

It does not.

The string `"False"` is non-empty, so it is truthy.

```
print(bool("False"))
```

Output:

```
True
```

Likewise:

```
print(bool("0"))
```

produces:

```
True
```

because `"0"` is still a non-empty string.

This is a very important concept when processing user input.

---

16. Type Conversion With `input()`

The `input()` function always returns a string.

Consider:

```
age = input("Enter your age: ")
```

If the user enters:

```
25
```

Python stores:

```
"25"
```

not:

```
25
```

Therefore, a common pattern is:

```
age = int(input("Enter your age: "))
```

Now the conversion happens immediately.

---

17. Converting Integer Input

```
age = int(input("Enter your age: "))

print(age + 1)
```

If the user enters:

```
25
```

Output:

```
26
```

The process is:

```
User enters "25"
       ↓
input() returns "25"
       ↓
int() converts "25"
       ↓
25
       ↓
25 + 1
       ↓
26
```

This pattern appears in many beginner Python programs.

---

18. Converting Float Input

Suppose a user enters a product price.

```
price = float(input("Enter price: "))

print(price)
```

If the user enters:

```
49.99
```

Python stores:

```
49.99 → float
```

Now you can perform numerical operations.

---

19. A Real World Shopping Example

Imagine a simple haas.dev course checkout.

```
course_price = float(input("Course price: "))
quantity = int(input("Quantity: "))

total = course_price * quantity

print("Total:", total)
```

If the user enters:

```
49.99
2
```

The program performs:

```
"49.99" → 49.99
"2"     → 2
```

Then:

```
49.99 × 2 = 99.98
```

The important part is not just the calculation.

The important part is transforming external text into data that the program can actually calculate with.

---

20. A Web Application Example

Imagine an API sends:

```
{
    "price": "49.99",
    "students": "120"
}
```

These values are strings.

Your Python application may need to perform calculations:

```
price = float(data["price"])
students = int(data["students"])
```

Now:

```
price    → float
students → int
```

You can calculate:

```
estimated_revenue = price * students
```

External data often needs validation and conversion before your application uses it.

---

21. Conversion Does Not Usually Change the Original Value

Consider:

```
age = "25"

number = int(age)

print(age)
print(number)
```

Output:

```
25
25
```

But the variables refer to values of different types:

```
age    → str
number → int
```

The conversion creates an integer value and assigns it to `number`.

If you want the original variable to hold the converted value:

```
age = int(age)
```

Now:

```
age → int
```

---

22. Conversion Chains

You can perform multiple conversions.

For example:

```
value = "25.5"

number = float(value)
number = int(number)

print(number)
```

Output:

```
25
```

The sequence is:

```
"25.5"
   ↓
25.5
   ↓
25
```

Be careful with multiple conversions because information can be lost.

The `.5` disappears when converting the float to an integer.

---

23. Information Loss During Conversion

Not every conversion preserves the exact original information.

Example:

```
number = int(9.8)

print(number)
```

Output:

```
9
```

The fractional component is gone.

Another example:

```
text = str(25)
```

The value is now text rather than a number.

Therefore, before converting data, ask:

> Will this conversion remove information that I still need?

This becomes especially important in real applications.

---

24. Invalid Conversions

Not every value can be converted successfully.

For example:

```
age = int("Hafsa")
```

Python cannot interpret `"Hafsa"` as an integer.

It raises:

```
ValueError
```

Similarly:

```
price = float("Python")
```

also raises a `ValueError`.

The conversion function must receive a compatible value.

---

25. `ValueError` vs `TypeError`

These two errors are especially useful to understand.

`ValueError`

The type is acceptable, but the actual value is not suitable.

Example:

```
int("hello")
```

Python received a string, but the contents cannot represent an integer.

`TypeError`

The operation or function received an inappropriate type.

Example:

```
"Age: " + 25
```

The operation attempts to combine a string and integer directly.

Understanding this difference will make debugging easier.

---

26. Safe Conversion With `try` and `except`

User input cannot always be trusted.

Suppose you write:

```
age = int(input("Enter your age: "))
```

What if the user enters:

```
twenty five
```

The conversion fails.

Later, when you learn exception handling, you can safely handle this:

```
try:
    age = int(input("Enter your age: "))
    print("Age:", age)
except ValueError:
    print("Please enter a valid number.")
```

The important idea is:

> External data should not automatically be assumed to be valid.

Exception handling will be covered separately in the Python series.

---

27. Converting Between Numeric Types

Common numeric conversions include:

```
int(10.8)
float(10)
```

Examples:

```
print(int(10.8))
print(float(10))
```

Output:

```
10
10.0
```

Remember:

```
int → whole number
float → decimal number
```

---

28. Converting Strings Carefully

String conversion is often safe when your goal is simply to represent a value as text.

```
print(str(25))
print(str(10.5))
print(str(True))
print(str(None))
```

Possible output:

```
25
10.5
True
None
```

But converting back requires the text to have an appropriate format.

For example:

```
int("25")
```

works.

But:

```
int("25.5")
```

does not.

If the text represents a decimal number, convert it to `float` first:

```
number = float("25.5")
```

If you specifically need an integer afterward:

```
number = int(float("25.5"))
```

Result:

```
25
```

---

29. Conversion With Expressions

You can convert values while performing an operation.

```
age = "25"

next_year = int(age) + 1

print(next_year)
```

Output:

```
26
```

Another example:

```
price = "49.99"
quantity = "3"

total = float(price) * int(quantity)

print(total)
```

Output:

```
149.97
```

This is useful when working with data received from outside the program.

---

30. Nested Conversion

You can put one conversion inside another:

```
value = int(float("25.9"))
```

The evaluation happens from the inside:

```
"25.9"
   ↓
float("25.9")
   ↓
25.9
   ↓
int(25.9)
   ↓
25
```

Although valid, do not create unnecessarily complicated conversion expressions.

Readable code is usually better:

```
value = float("25.9")
value = int(value)
```

---

31. Collection Type Conversion Preview

Python can also convert between collection types.

For example:

```
numbers = [1, 2, 3]

numbers_tuple = tuple(numbers)

print(numbers_tuple)
```

Output:

```
(1, 2, 3)
```

Or:

```
numbers = (1, 2, 3)

numbers_list = list(numbers)

print(numbers_list)
```

Output:

```
[1, 2, 3]
```

You can also create a set:

```
numbers = [1, 2, 2, 3]

unique_numbers = set(numbers)

print(unique_numbers)
```

Output contains unique values.

Lists, tuples, sets, and dictionaries will be covered in depth later.

---

32. Type Conversion in a haas.dev Learning System

Imagine haas.dev receives a learner's progress from a form:

```
progress = "87.5"
completed_lessons = "32"
```

These values arrive as strings.

Convert them:

```
progress = float(progress)
completed_lessons = int(completed_lessons)
```

Now the application can make decisions:

```
if progress >= 80:
    print("Strong progress")
```

And calculations:

```
remaining_lessons = 40 - completed_lessons
```

Without correct conversion, these operations may fail or behave differently than expected.

---

33. Type Conversion and Validation Are Different

These concepts are related but not identical.

Conversion

Changes the type:

```
age = int("25")
```

Validation

Checks whether the value is acceptable.

For example:

```
age = 25
```

The conversion succeeded, but you might still need to validate:

```
Is the age positive?
Is it within an expected range?
```

A value can be successfully converted and still be invalid for your application's rules.

For example:

```
age = int("-500")
```

Conversion succeeds.

But your application may reject the value because it does not make sense for the specific context.

---

34. Conversion and User Input Pipeline

A useful real-world mental model is:

```
External Data
     ↓
Receive
     ↓
Inspect
     ↓
Convert
     ↓
Validate
     ↓
Use
```

For example:

```
User enters "25"
        ↓
input()
        ↓
"25" as str
        ↓
int()
        ↓
25 as int
        ↓
validation
        ↓
application logic
```

This pattern appears throughout professional software development.

---

35. Common Beginner Mistakes

Mistake 1: Forgetting That `input()` Returns Strings

Incorrect:

```
age = input("Age: ")
print(age + 1)
```

Correct:

```
age = int(input("Age: "))
print(age + 1)
```

Mistake 2: Assuming `"10"` Is a Number

It is text:

```
"10"
```

To convert:

```
int("10")
```

Mistake 3: Expecting `int()` to Round

```
int(7.9)
```

produces:

```
7
```

It does not produce `8`.

Mistake 4: Assuming `bool("False")` Is False

```
bool("False")
```

produces:

```
True
```

because the string is not empty.

Mistake 5: Converting Without Considering Data Loss

```
int(12.99)
```

produces:

```
12
```

The decimal portion is lost.

Mistake 6: Converting Before Understanding the Data

Do not blindly write:

```
int(data)
```

Ask:

- What type is `data`?
- What does it represent?
- Is its format valid?
- Could conversion fail?
- Will conversion lose information?

---

36. Best Practices

1. Convert at the boundary

When receiving external data, convert it near the point where you receive it.

Instead of carrying `"25"` throughout your program:

```
age = int(input("Age: "))
```

Now the rest of your application can work with an integer.

---

2. Choose the correct target type

Use:

```
int()
```

for whole numbers.

Use:

```
float()
```

for decimal numbers.

Use:

```
str()
```

for text.

Use:

```
bool()
```

when you intentionally need truthiness conversion.

---

3. Validate external data

Successful conversion does not guarantee valid application data.

Check your application's rules after conversion.

---

4. Be careful with information loss

Before converting:

```
float → int
```

consider whether the decimal component matters.

---

5. Keep conversions readable

Instead of:

```
result = int(float(input("Number: ")))
```

you may prefer:

```
value = input("Number: ")
value = float(value)
value = int(value)
```

When the process matters, readable code is easier to debug.

---

37. Quick Reference Table

Conversion	Example	Result
String → Integer	`int("25")`	`25`
String → Float	`float("25.5")`	`25.5`
Integer → Float	`float(25)`	`25.0`
Float → Integer	`int(25.9)`	`25`

Integer → String	`str(25)`	`"25"`
Float → String	`str(25.5)`	`"25.5"`
Boolean → Integer	`int(True)`	`1`
Integer → Boolean	`bool(1)`	`True`
Empty String → Boolean	`bool("")`	`False`
Non empty String → Boolean	`bool("Python")`	`True`

---

38. Conversion Flow Cheat Sheet

```
"25"
 ↓
int()
 ↓
25
```

```
"25.5"
 ↓
float()
 ↓
25.5
```

```
25
 ↓
str()
 ↓
"25"
```

```
1
 ↓
bool()
 ↓
True
```

```
0
 ↓
bool()
 ↓
False
```

---

39. Mini Quiz

Question 1

What is the result?

```
int("50")
```

A. `"50"`

B. `50` as an integer

C. `50.0`

D. Error

Answer: B

Question 2

What is the result?

```
float("10.5")
```

A. String

B. Integer

C. Float

D. Boolean

Answer: C

Question 3

What does this produce?

```
int(8.9)
```

A. `8`

B. `9`

C. `8.9`

D. Error

Answer: A

Question 4

What does this produce?

```
bool("")
```

A. `True`

B. `False`

C. `None`

D. Error

Answer: B

Question 5

What does `input()` return?

A. Integer

B. Float

C. String

D. Boolean

Answer: C

Question 6

What happens here?

```
int("Python")
```

A. Returns `0`

B. Returns `None`

C. Converts it to a boolean

D. Raises `ValueError`

Answer: D

---

40. 5 Minute Challenge

Build a small **Course Price Calculator**.

Ask the user for:

```
Course price:
Number of courses:
```

Both values will initially arrive as strings.

Convert:

- course price → `float`
- number of courses → `int`

Then calculate:

```
total = price × number of courses
```

Print:

```
Total: ...
```

Bonus

Also ask for a discount percentage.

For example:

```
Course price: 50
Number of courses: 3
Discount: 10
```

Calculate the final price after the discount.

Think carefully about which values should be `str`, `int`, and `float`.

---

41. Practice Exercises

Easy

1. Convert `"100"` into an integer.
2. Convert `"25.5"` into a float.
3. Convert `50` into a string.
4. Convert `10` into a float.
5. Convert `1` into a boolean.
6. Convert `0` into a boolean.

7. Check the type of every converted value.

Medium

1. Ask the user for two numbers.
2. Convert both inputs into integers.
3. Add them.
4. Subtract them.
5. Multiply them.
6. Divide them.
7. Print the results.

Hard

Create a simple student result calculator.

Ask for:

```
Student name
Marks in subject 1
Marks in subject 2
Marks in subject 3
Total possible marks
```

Remember:

- name should remain a string
- marks should be numeric
- total possible marks should be numeric

Calculate:

```
total marks
percentage
```

Then determine whether the student passed based on a percentage threshold you choose.

Handle invalid numerical input using `try` and `except`.

---

42. Interview Questions

1. What is type conversion?

Type conversion is changing a value from one data type to another.

2. How do you convert a string to an integer?

```
number = int("25")
```

3. How do you convert an integer to a float?

```
number = float(25)
```

4. How do you convert a number to a string?

```
text = str(25)
```

5. What does `bool()` do?

It converts a value to a boolean according to Python's truthiness rules.

6. What is the difference between `int(9.9)` and `round(9.9)`?

```
int(9.9)
```

produces `9`.

```
round(9.9)
```

produces `10`.

`int()` removes the fractional portion toward zero, while `round()` performs rounding.

7. What happens when `int("Python")` is executed?

Python raises a `ValueError` because `"Python"` cannot be interpreted as an integer.

8. Why is type conversion important when using `input()`?

Because `input()` returns a string regardless of what the user enters. Numerical input must be converted before performing numerical operations.

9. Does successful type conversion mean the data is valid?

No. Conversion only changes the representation/type. Application-specific validation may still be required.

10. What is a truthy value?

A value that Python treats as `True` in a boolean context.

For example:

```
bool("Python")
```

returns `True`.

---

43. Key Takeaways

1. **Type conversion changes a value from one type to another.**
2. **`int()` creates integers from compatible values.**

3. **`float()` creates floating-point numbers.**
4. **`str()` creates strings.**
5. **`bool()` creates boolean values according to truthiness rules.**
6. **`input()` always returns a string.**
7. **External data often needs conversion before calculations.**
8. **Not every conversion is possible.**
9. **Invalid conversions can raise `ValueError`.**
10. **Operations involving incompatible types can raise `TypeError`.**
11. **Converting a float to an integer can discard information.**
12. **`bool("False")` is `True` because the string is non-empty.**
13. **Conversion and validation are separate steps.**
14. **Convert external data deliberately rather than blindly.**
15. **Good type handling makes programs easier to debug and maintain.**

---

44. Final Mental Model

When your Python program receives a value, think:

```
What type did I receive?
          ↓
What type do I need?
          ↓
Can the value be converted safely?
          ↓
Convert it
          ↓
Validate it
          ↓
Use it
```

For example:

```
User enters "25"
        ↓
input()
        ↓
str
        ↓
int()
```

```
        ↓
25
        ↓
validate
        ↓
use in calculations
```

Type conversion may look like a small Python concept, but it becomes a major part of real applications.

Forms, APIs, files, databases, JSON, command-line programs, dashboards, and backend systems constantly move data between different representations.

Learning to recognize, convert, and validate types is therefore an essential programming skill.

---

haas.dev

https://dev-roast-app.vercel.app

Visit haas.dev for more resources and guides.