Author: PyShaala
Date: August 13, 2025
YouTube Channel: PyShaala

📚 What are Integers and Floats?

In Python, numbers are mainly divided into two types:

1. **Integers (`int`)** – Whole numbers without a decimal point.
2. **Floats (`float`)** – Numbers with a decimal point.

These are the most commonly used numeric data types in programming.
They are essential for performing mathematical operations, measurements, and calculations.

Integers (`int`)

Integers are whole numbers, positive or negative, without a decimal point. They can be of unlimited size, limited only by the available memory.

- **Characteristics:**
  - Represent whole numbers (e.g., -2, 0, 100, 1000000000).
  - Do not have a fractional part.
  - Can be very large (limited by memory).

Floats (`float`)

Floats, also known as floating-point numbers, are numbers with a decimal point or numbers written in exponential form (e.g., `2.5e4` for 2.5 * 10^4). They represent real numbers.

- **Characteristics:**
  - Represent numbers with a fractional part (e.g., 3.14, -0.001, 1.5e-10).
  - Have limited precision due to the way they are stored in computer memory (typically using the IEEE 754 standard). This can sometimes lead to small rounding errors in calculations.

Key Differences:

Feature	Integer (`int`)	Float (`float`)
Represents	Whole numbers	Numbers with decimal points (real numbers)
Decimal	No	Yes
Precision	Arbitrary (limited by memory)	Limited
Syntax	`10` , `-5` , `0`	`3.14` , `-0.001` , `1.2e5`

📝 Syntax

Creating Integers

Integers are created by simply writing the whole number.

```
# Positive integer
positive_int = 100

# Negative integer
negative_int = -50

# Zero
zero_int = 0

# Large integer
large_int = 12345678901234567890
```

Creating Floats

Floats are created by writing a number with a decimal point or using scientific notation (e.g., `e` or `E` followed by an exponent).

```
# Float with a decimal point
simple_float = 3.14

# Negative float
negative_float = -0.001

# Float equal to a whole number (still a float type)
whole_number_float = 5.0

# Float in scientific notation (e notation)
scientific_float = 1.5e2  # Represents 1.5 * 10^2 = 150.0
small_scientific_float = 2.3e-3 # Represents 2.3 * 10^-3 = 0.0023
```

In [29]:
```
# Examples of creating and checking types of integers and floats
# Integers
int1 = 42
```

```
int2 = -100
int3 = 0
int4 = 98765432109876543210

print(f"Value: {int1}, Type: {type(int1)}")
print(f"Value: {int2}, Type: {type(int2)}")
print(f"Value: {int3}, Type: {type(int3)}")
print(f"Value: {int4}, Type: {type(int4)}")
```

```
Value: 42, Type: <class 'int'>
Value: -100, Type: <class 'int'>
Value: 0, Type: <class 'int'>
Value: 98765432109876543210, Type: <class 'int'>
```

In [30]:
```
# Floats
float1 = 3.14159
float2 = -0.05
float3 = 10.0
float4 = 6.022e23 # Avogadro's number in scientific notation
float5 = 1.6e-19  # Charge of an electron

print(f"Value: {float1}, Type: {type(float1)}")
print(f"Value: {float2}, Type: {type(float2)}")
print(f"Value: {float3}, Type: {type(float3)}")
print(f"Value: {float4}, Type: {type(float4)}")
print(f"Value: {float5}, Type: {type(float5)}")
```

```
Value: 3.14159, Type: <class 'float'>
Value: -0.05, Type: <class 'float'>
Value: 10.0, Type: <class 'float'>
Value: 6.022e+23, Type: <class 'float'>
Value: 1.6e-19, Type: <class 'float'>
```

➕➖✖➗ Arithmetic Operations

Python supports standard arithmetic operations between integers and floats. When an operation involves both an integer and a float, the integer is usually converted to a float before the operation is performed, resulting in a float.

Here are the basic arithmetic operators:

- `+` : Addition
- `-` : Subtraction
- `*` : Multiplication
- `/` : Division (always returns a float)
- `//` : Floor division (returns an integer, discarding the fractional part)
- `%` : Modulo (returns the remainder of the division)
- `**` : Exponentiation (raising to a power)

In [40]:
```
# Examples of arithmetic operations

# Integers
```

```
a = 10
b = 3

print(f"{a} + {b} = {a + b}")
print(f"{a} - {b} = {a - b}")
print(f"{a} * {b} = {a * b}")
print(f"{a} / {b} = {a / b}")   # Float division
print(f"{a} // {b} = {a // b}") # Floor division
print(f"{a} % {b} = {a % b}")   # Modulo
print(f"{a} ** {b} = {a ** b}") # Exponentiation
```

```
10 + 3 = 13
10 - 3 = 7
10 * 3 = 30
10 / 3 = 3.3333333333333335
10 // 3 = 3
10 % 3 = 1
10 ** 3 = 1000
```

In [32]:
```
# Floats
x = 10.5
y = 2.5

print(f"{x} + {y} = {x + y}")
print(f"{x} - {y} = {x - y}")
print(f"{x} * {y} = {x * y}")
print(f"{x} / {y} = {x / y}")
```

```
10.5 + 2.5 = 13.0
10.5 - 2.5 = 8.0
10.5 * 2.5 = 26.25
10.5 / 2.5 = 4.2
```

In [33]:
```
# Mixed types (Integer and Float)
c = 5
d = 2.0

print(f"{c} + {d} = {c + d}") # Result is a float
print(f"{c} * {d} = {c * d}") # Result is a float
print(f"{a} / {y} = {a / y}") # Integer divided by float, result is float
```

```
5 + 2.0 = 7.0
5 * 2.0 = 10.0
10 / 2.5 = 4.0
```

In [34]:
```
9 // 2.0
```

Out[34]:
```
4.0
```

1 2 3 4 Common Functions

Python provides several built-in functions that are useful when working with integers and floats. Here are a few common ones:

- `abs(x)`: Returns the absolute value of a number `x`.

- `round(x, n)` : Rounds a number `x` to `n` decimal places. If `n` is omitted, it rounds to the nearest integer.
- `int(x)` : Converts a number or string `x` to an integer.
- `float(x)` : Converts a number or string `x` to a float.

In [35]:
```
# Examples of common functions
# abs()
print(f"Absolute value of -15: {abs(-15)}")
print(f"Absolute value of -7.8: {abs(-7.8)}")
```

```
Absolute value of -15: 15
Absolute value of -7.8: 7.8

Absolute value of -7.8: 7.8
```

- `round(x, n)` : Rounds a number `x` to `n` decimal places. If `n` is omitted, it rounds to the nearest integer.

In [36]:
```
# round()
print(f"Round 3.14159 to nearest integer: {round(3.14159)}")
print(f"Round 3.14159 to 2 decimal places: {round(3.14159, 2)}")
print(f"Round 15.478 to 1 decimal place: {round(15.478, 1)}")
print(f"Round 15.678 to nearest integer: {round(15.678)}")
```

```
Round 3.14159 to nearest integer: 3
Round 3.14159 to 2 decimal places: 3.14
Round 15.478 to 1 decimal place: 15.5
Round 15.678 to nearest integer: 16
```

- `int(x)` : Converts a number or string `x` to an integer.
- `float(x)` : Converts a number or string `x` to a float.

In [37]:
```
# Type conversion
integer_num = 10
float_num = 5.7
string_int = "123"
string_float = "45.67"

print(f"Convert float {float_num} to integer: {int(float_num)}")
print(f"Convert string '{string_int}' to integer: {int(string_int)}")
# print(f"Convert string '{string_float}' to integer: {int(string_float)}") # This

print(f"Convert integer {integer_num} to float: {float(integer_num)}")
print(f"Convert string '{string_float}' to float: {float(string_float)}")
print(f"Convert string '{string_int}' to float: {float(string_int)}")
```

```
Convert float 5.7 to integer: 5
Convert string '123' to integer: 123
Convert integer 10 to float: 10.0
Convert string '45.67' to float: 45.67
Convert string '123' to float: 123.0
```

✨ Best Practices

Following these best practices can help you work effectively and avoid common pitfalls when dealing with integers and floats in Python:

For Integers

- **Use Integers for Counting and Exact Values:** Integers are best suited for representing counts, indices, whole quantities, or any value that must be exact.
- **Be Mindful of Division:** Remember that standard division (`/`) always results in a float, even if the numbers divide evenly. Use floor division (`//`) if you explicitly need an integer result from division.
- **Large Integers are Fine:** Don't shy away from using large integers; Python handles them efficiently.

For Floats

- **Understand Floating-Point Precision:** Be aware that floats have limited precision. Avoid comparing floating-point numbers directly for exact equality (`==`). Instead, check if the absolute difference between them is within a small tolerance (epsilon).
- **Use Decimals for Financial Calculations:** For applications requiring exact decimal representation, such as financial calculations, use the `Decimal` type from the `decimal` module. This avoids the precision issues inherent in standard floats.
- **Consider Rounding When Displaying:** When displaying floating-point numbers, consider rounding them to a reasonable number of decimal places for better readability.
- **Type Conversion When Necessary:** Explicitly convert strings to floats using `float()` when processing numerical input from users or files.

General

- **Choose the Right Type:** Select `int` or `float` based on whether you need to represent whole numbers or numbers with decimal places.
- **Use Meaningful Variable Names:** Give your numeric variables descriptive names (e.g., `total_count` , `average_price` , `temperature_celsius`).
- **Convert Types Explicitly:** When performing operations that require specific types (e.g., integer division), explicitly convert types using `int()` , `float()` , etc., for clarity.
- **Use floats for continuous measurements:** continuous values refer to numerical data that can take on any value within a given range, including fractions and decimals. `These values are typically measured rather than counted` , and there are no clear, distinct gaps between potential values.

✅ Do's and ❌ Don'ts

Here are some important things to remember when working with integers and floats:

✅ Do's

- **Do** use `int` for counting and whole number quantities.
- **Do** use `float` for measurements and numbers with decimal parts.
- **Do** use floor division (`//`) when you need an integer result from division.
- **Do** use `try-except` blocks when converting user input to `int` or `float` to handle potential `ValueError`.
- **Do** use the `decimal` module for financial or other applications requiring exact decimal precision.
- **Do** round floats for display purposes when full precision is not necessary.

❌ Don'ts

- **Don't** compare floats for exact equality (`==`) directly.
- **Don't** forget that standard division (`/`) always results in a float.
- **Don't** assume user input for numbers will always be valid; always plan for type conversion errors.

🌍 Real-Life Examples

Integers and floats are used extensively in real-world programming to represent various types of numerical data. Here are a few examples demonstrating their application.

In [38]:
```python
# Example 1: Temperature Conversion

# This program converts a temperature from Celsius to Fahrenheit.

# Get temperature in Celsius from user (input is a string)
celsius_str = input("Enter temperature in Celsius: ")

# Convert the input string to a float
try:
    celsius = float(celsius_str)

    # Formula to convert Celsius to Fahrenheit: F = (C * 9/5) + 32
    fahrenheit = (celsius * 9/5) + 32

    # Print the result, rounded to two decimal places
    print(f"{celsius}°C is equal to {fahrenheit:.3f}°F")

except ValueError:
    print("Invalid input. Please enter a valid number for temperature.")
```

```
55.0°C is equal to 131.000°F
```

In [39]:
```python
# Example 2: Calculating Average Score

# This program calculates the average of a list of test scores.

# List of test scores (integers)
test_scores = [85, 90, 78, 92, 88]

# Calculate the total sum of scores
total_score = sum(test_scores) # sum() works with lists of numbers

# Calculate the number of scores (an integer)
number_of_scores = len(test_scores)

# Calculate the average score (will be a float if total is not perfectly divisible)
average_score = total_score / number_of_scores

# Print the results
print(f"Test Scores: {test_scores}")
print(f"Total Score: {total_score}")
print(f"Number of Scores: {number_of_scores}")
print(f"Average Score: {average_score:.2f}") # Display average with 2 decimal place

# Example demonstrating integer vs float division
print(f"Total score divided by 5 (float division): {total_score / 5}")
print(f"Total score divided by 5 (floor division): {total_score // 5}")
```

```
Test Scores: [85, 90, 78, 92, 88]
Total Score: 433
Number of Scores: 5
Average Score: 86.60
Total score divided by 5 (float division): 86.6
Total score divided by 5 (floor division): 86
```

Practice Exercises

Test your understanding of Python integers and floats with the following exercises. Write your code in the cells below each problem description.

Exercise 1: Integer and Float Creation

1. Create an integer variable `students_count` and assign it the value 35.
2. Create a float variable `temperature` and assign it the value 24.7.
3. Create a large integer variable `world_population` (use a value around 8 billion).
4. Create a float variable `pi_value` using scientific notation (e.g., `3.14159e0`).
5. Print the value and type of each variable.

Exercise 2: Arithmetic Operations

1. Start with `students_count = 35` and `temperature = 24.7`.

2. Create a new integer variable `new_students` and assign it 5. Add `new_students` to `students_count` and store the result in `total_students`. Print `total_students`.
3. Subtract 2.5 from `temperature` and store the result in `cooler_temperature`. Print `cooler_temperature`.
4. Multiply `temperature` by 1.8 and add 32 (Fahrenheit conversion). Store the result in `temp_fahrenheit`. Print `temp_fahrenheit`.
5. Divide `total_students` by 3 and print the result (note the type).
6. Use floor division to divide `total_students` by 3 and print the result.
7. Find the remainder when `students_count` is divided by 4. Print the result.

Exercise 3: Using `abs()` and `round()`

1. Create a float variable `balance` with a value of -150.75. Use `abs()` to find its absolute value and print it.
2. Create a float variable `long_float = 123.456789`. Use `round()` to round it to 2 decimal places and print the result.
3. Use `round()` on `long_float` without specifying decimal places and print the result.
4. Convert `long_float` to an integer using `int()` and print the result.
5. Convert the integer `students_count` (from Exercise 1) to a float using `float()` and print the result.

✨Summary

1. What are Integers and Floats?
2. Syntax
3. ➕➖✖️➗ Arithmetic Operations
4. 🔢 Common Functions
5. ✨ Best Practices
6. ✅ Do's and ❌ Don'ts
7. 🌎 Real-Life Examples
8. 🏋️ Practice Exercises

🏋️ Interview Question

`For Answer please visit our youtube channel short video section or Instagram page`

YouTube Channel: PyShaala

🟢 Basic Level

- What is the difference between int and float in Python?

- How does Python handle very large integers?
- How do you convert a float to an integer? What happens to the decimal part?
- How do you check if a variable is an integer or float?
- How do you convert a string "123.45" to a float in Python?
- How do you round a float value to 2 decimal places in Python?
- What happens if you divide two integers in Python?
- What does the // operator do in Python?
- What is the result of 5 / 2 and 5 // 2?
- How do you represent negative integers and floats in Python?

🟡 Intermediate Level

- How does Python store floating-point numbers internally?
- Why does 0.1 + 0.2 not exactly equal 0.3 in Python?
- How do you handle precision issues with floats in Python?
- How do you check if two floats are equal considering floating-point precision?
- What is the use of Decimal and Fraction classes in Python?
- How do you format a float for display with specific decimal places?
- What is integer overflow, and does Python have it?
- What's the difference between math.floor() and int() when converting floats?
- How do you check if a float is an integer value without converting it?
- What is the output of int(True) and float(False)?

🔴 Advanced / Tricky Level

- How does Python handle division by zero for integers and floats?
- What is the result of float('inf') - float('inf')?
- How do you check if a number is NaN (Not a Number)?
- Why might comparing floats directly with == cause bugs?
- How can you represent numbers in scientific notation in Python?
- How do you round a float to the nearest even number?
- What happens if you convert a very large float to an integer?
- What is the difference between math.trunc() and round()?
- How does Python handle negative float floor division?
- How to avoid precision loss when adding large and small float numbers together?