Python Basics: Numbers and Math

Python Basics: Numbers and Math

Computers use numbers to represent:

- 📄 Text
- 🎨 Images
- 🎶 Music
- 📽️ Videos

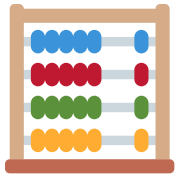
…and everything else!

Python Basics: Numbers and Math

Computers use numbers to represent:

- 📄 Text
- 🎨 Images
- 🎶 Music
- 📽️ Videos

…and everything else!

&


Python Basics: Numbers and Math

Computers use numbers to represent:

- 📄 Text
- 🎨 Images
- 🎶 Music
- 📽️ Videos

…and everything else!

Python Basics: Numbers and Math

Computers use numbers to represent:

- 📄 Text
- 🎨 Images
- 🎶 Music
- 📽️ Videos

…and everything else!

Python Basics: Numbers and Math

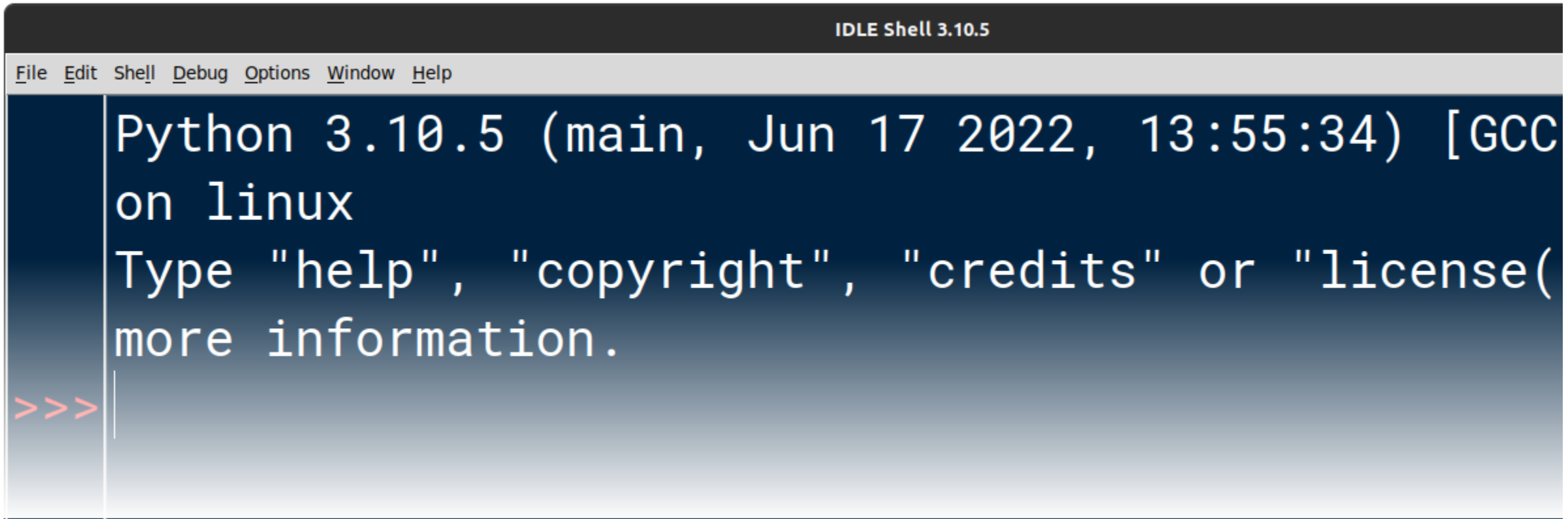
In this course you'll:

- Get an overview of the numeric types in Python
- Create integers, floats, and complex numbers
- Perform arithmetic operations
- Learn about the floating-point representation error
- Work with math functions and number methods
- Format and display numbers as strings

Python Basics: Numbers and Math

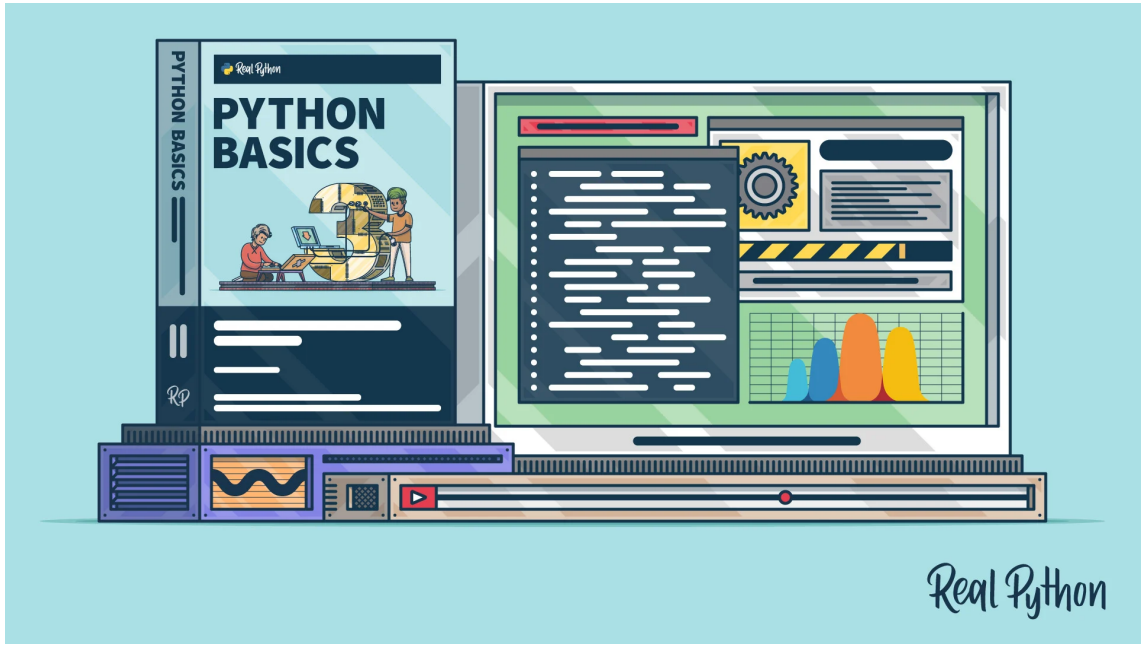
What you'll need:

- **IDLE: I**ntegrated **D**evelopment and **L**earning **E**nvironment

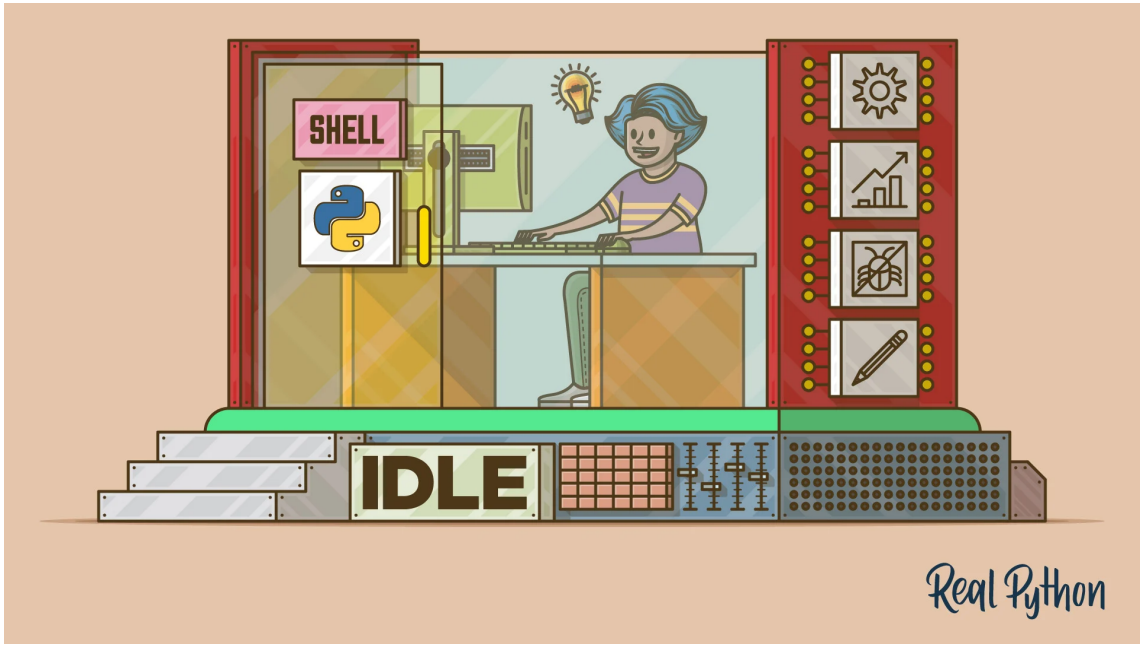


Python Basics: Numbers and Math

What you'll need:



Python Basics: Setting Up Python



Starting With Python IDLE

Table of Contents

Table of Contents

Numeric Types in Python

Numeric Types in Python

Built-in Types:

Numeric Types in Python

Built-in Types:

- **Integers:** `-3`, `0`, `42`

Numeric Types in Python

Built-in Types:

- **Integers (int):** `-3`, `0`, `42`

Numeric Types in Python

Built-in Types:

- **Integers (int):** `-3`, `0`, `42`
- **Floating-Point Numbers:** `-2.72`, `3.14`

Numeric Types in Python

Built-in Types:

- **Integers (int):** `-3`, `0`, `42`
- **Floating-Point Numbers (float):** `-2.72`, `3.14`

Numeric Types in Python

Built-in Types:

- **Integers (int):** `-3`, `0`, `42`
- **Floating-Point Numbers (float):** `-2.72`, `3.14`
- **Complex Numbers:** `-0.744 + 0.132j`

Numeric Types in Python

Built-in Types:

- **Integers (int):** `-3`, `0`, `42`
- **Floating-Point Numbers (float):** `-2.72`, `3.14`
- **Complex Numbers (complex):** `-0.744 + 0.132j`

Numeric Types in Python

Built-in Types:

- ± Integers (int): `-3`, `0`, `42`
- ± Floating-Point Numbers (float): `-2.72`, `3.14`
- ± Complex Numbers (complex): `-0.744 + 0.132j`

Numeric Types in Python

Built-in Types:

- **± Integers (int):** `-3`, `0`, `42`
  - Unlimited, arbitrary precision
- **± Floating-Point Numbers (float):** `-2.72`, `3.14`
- **± Complex Numbers (complex):** `-0.744 + 0.132j`

Numeric Types in Python

Built-in Types:

- **± Integers (int):** `-3`, `0`, `42`
  - Unlimited, arbitrary precision
- **± Floating-Point Numbers (float):** `-2.72`, `3.14`
  - Smallest: $2.2250738585072014 \times 10^{-308}$
  - Minimum: $-1.7976931348623157 \times 10^{308}$
  - Maximum: $1.7976931348623157 \times 10^{308}$
- **± Complex Numbers (complex):** `-0.744 + 0.132j`

Numeric Types in Python

Built-in Types:

- **± Integers (int):** `-3`, `0`, `42`
  - Unlimited, arbitrary precision
- **± Floating-Point Numbers (float):** `-2.72`, `3.14`
  - Smallest: $2.2250738585072014 \times 10^{-308}$
  - Minimum: $-1.7976931348623157 \times 10^{308}$
  - Maximum: $1.7976931348623157 \times 10^{308}$
- **± Complex Numbers (complex):** `-0.744 + 0.132j`
  - Real and imaginary parts are floating-point numbers

Numeric Types in Python

Built-in Types:

- **± Integers (int):** `-3`, `0`, `42`
  - **Boolean (bool):** `True` ≡ `1`, `False` ≡ `0`
  - Unlimited, arbitrary precision
- **± Floating-Point Numbers (float):** `-2.72`, `3.14`
  - Smallest: $2.2250738585072014 \times 10^{-308}$
  - Minimum: $-1.7976931348623157 \times 10^{308}$
  - Maximum: $1.7976931348623157 \times 10^{308}$
- **± Complex Numbers (complex):** `-0.744 + 0.132j`
  - Real and imaginary parts are floating-point numbers

Numeric Types in Python

Built-in Types:

- **± Integers (int):** `-3`, `0`, `42`
  - ~~**Boolean (bool):** `True` == `1`, `False` == `0`~~
  - Unlimited, arbitrary precision
- **± Floating-Point Numbers (float):** `-2.72`, `3.14`
  - Smallest: $2.2250738585072014 \times 10^{-308}$
  - Minimum: $-1.7976931348623157 \times 10^{308}$
  - Maximum: $1.7976931348623157 \times 10^{308}$
- **± Complex Numbers (complex):** `-0.744 + 0.132j`
  - Real and imaginary parts are floating-point numbers

Numeric Types in Python

Built-in Types:

- ± Integers (int): `-3`, `0`, `42`
  - ~~Boolean (bool): `True` == `1`, `False` == `0`~~
  - Unlimited, arbitrary precision
- ± Floating-Point Numbers (float): `-2.72`, `3.14`
  - Smallest: $2.2250738585072014 \times 10^{-308}$
  - Minimum: $-1.7976931348623157 \times 10^{308}$
  - Maximum: $1.7976931348623157 \times 10^{308}$
- ± Complex Numbers (complex): `-0.744 + 0.132j`
  - Real and imaginary parts are floating-point numbers

Numeric Types in Python

Built-in Types:

- ✅ ± Integers (int): `-3`, `0`, `42`
  - ~~Boolean (bool): `True` == `1`, `False` == `0`~~
  - Unlimited, arbitrary precision
- ❌ ± Floating-Point Numbers (float): `-2.72`, `3.14`
  - Smallest: $2.2250738585072014 \times 10^{-308}$
  - Minimum: $-1.7976931348623157 \times 10^{308}$
  - Maximum: $1.7976931348623157 \times 10^{308}$
- ± Complex Numbers (complex): `-0.744 + 0.132j`
  - Real and imaginary parts are floating-point numbers

Numeric Types in Python

Built-in Types:

- ✅ ± Integers (int): `-3`, `0`, `42`
  - ~~Boolean (bool): `True` == `1`, `False` == `0`~~
  - Unlimited, arbitrary precision
- ❌ ± Floating-Point Numbers (float): `-2.72`, `3.14`
  - Smallest: $2.2250738585072014 \times 10^{-308}$
  - Minimum: $-1.7976931348623157 \times 10^{308}$
  - Maximum: $1.7976931348623157 \times 10^{308}$
- ± Complex Numbers (complex): `-0.744 + 0.132j`
  - Real and imaginary parts are floating-point numbers

Numeric Types in Python

Built-in Types:

- ✅ ± Integers (int): `-3`, `0`, `42`
  - ~~Boolean (bool): `True` == `1`, `False` == `0`~~
  - Unlimited, arbitrary precision
- ❌ ± Floating-Point Numbers (float): `-2.72`, `3.14`
  - Smallest: $2.2250738585072014 \times 10^{-308}$
  - Minimum: $-1.7976931348623157 \times 10^{308}$
  - Maximum: $1.7976931348623157 \times 10^{308}$
- ± Complex Numbers (complex): `-0.744 + 0.132j`
  - Real and imaginary parts are floating-point numbers

$3.99 ≡ 399

Numeric Types in Python

Additional Types:

Numeric Types in Python

Additional Types:

- **Decimal:** `decimal.Decimal("0.33")`

Numeric Types in Python

Additional Types:

- **Decimal:** `decimal.Decimal("0.33333333333333333333333333")`
  - Arbitrary but finite precision (28 digits by default)

Numeric Types in Python

Additional Types:

- **Decimal:** `decimal.Decimal("0.33333333333333333333333333")`
  - Arbitrary but finite precision (28 digits by default)
  - Can’t interoperate with floating-point numbers

Numeric Types in Python

Additional Types:

- **Decimal:** `decimal.Decimal("0.33333333333333333333333333")`
  - Arbitrary but finite precision (28 digits by default)
  - Can't interoperate with floating-point numbers
- **Fraction:** `fractions.Fraction(1, 3)`
  - Can interoperate with floating-point numbers

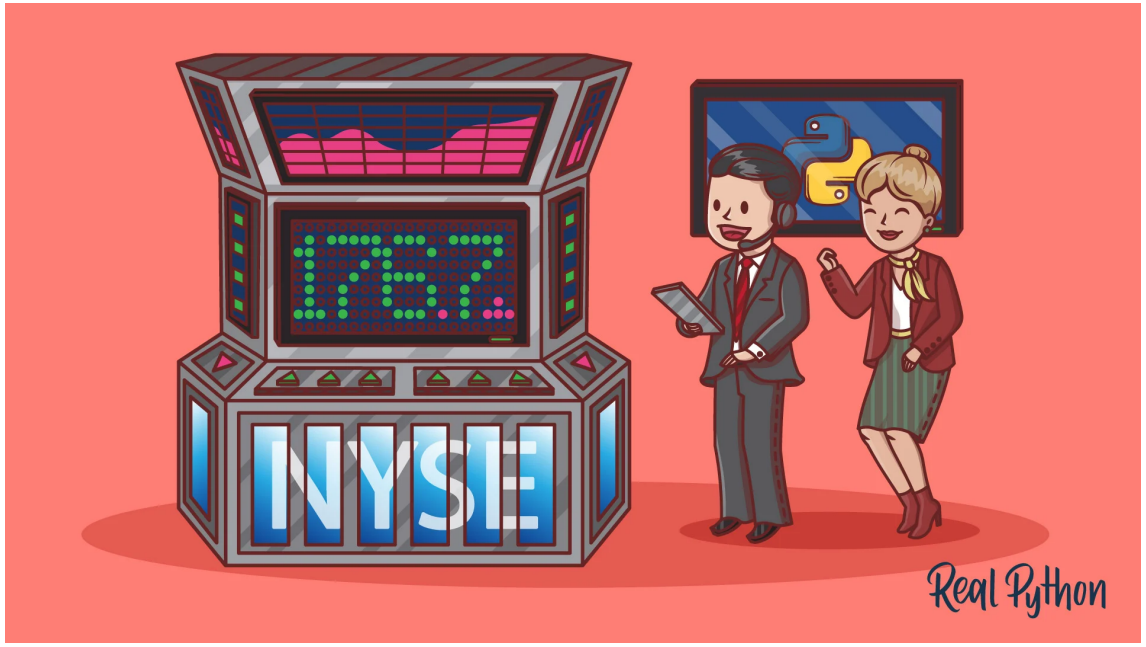
Numeric Types in Python

Additional Types:

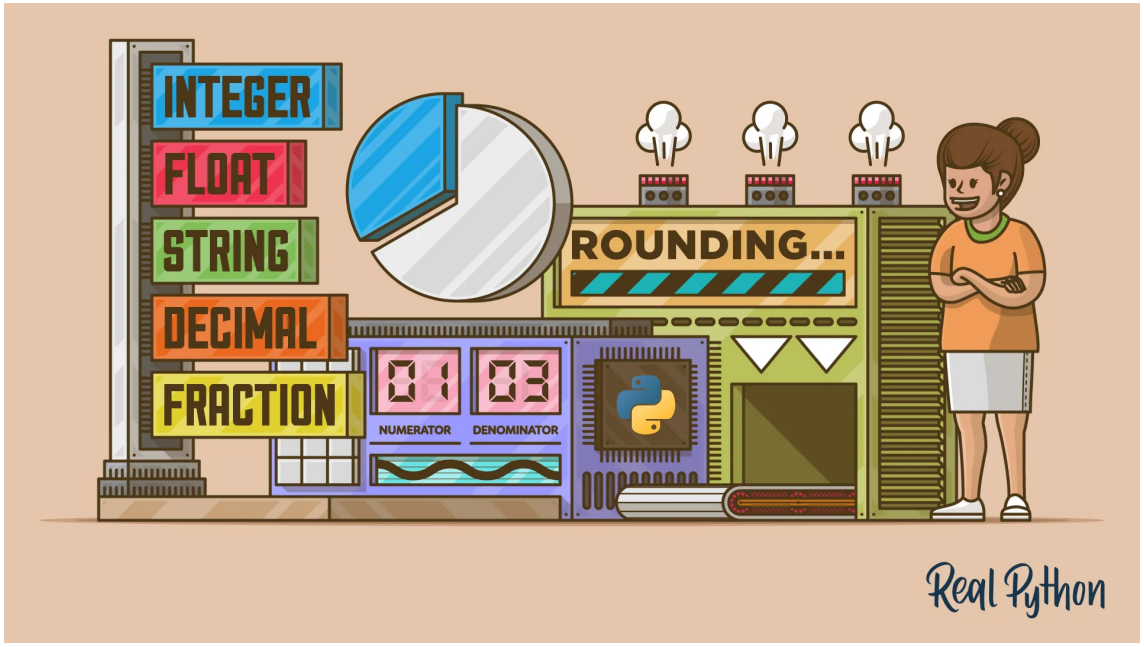
- **Decimal:** `decimal.Decimal("0.33333333333333333333333333")`
  - Arbitrary but finite precision (28 digits by default)
  - Can't interoperate with floating-point numbers
- **Fraction:** `fractions.Fraction(1, 3)`
  - Can interoperate with floating-point numbers
  - Infinite precision: ⅓ = 0.333…

Numeric Types in Python

Additional Types:



How to Round Numbers in Python



Representing Rational Numbers With Python Fractions

Table of Contents

Table of Contents

Integers

Integer Literals:

- Decimal: `42`
- Binary: `0b101010`
- Hexadecimal: `0x2a`
- Octal: `0o52`

Integers

Integer Literals:

- **Decimal:** `42`
- **Binary:** `0b101010`
- **Hexadecimal:** `0x2a`
- **Octal:** `0o52`

The `int()` Function:

- `int()` → `0`
- `int("42")` → `42`
- `int(3.99)` → `3`
- `int("101010", 2)` → `42`

Table of Contents

Table of Contents

Floating-Point Numbers

Floating-Point Numbers

Floating-Point Literals:

- `-4.2`
- `0.42`
- `42.0`
- `42_000_000.0`

Floating-Point Numbers

Floating-Point Literals:

- `-4.2`
- `.42`
- `42.`
- `42_000_000.`

Floating-Point Numbers

Floating-Point Literals:

- `-4.2`
- `.42`
- `42.`
- `42_000_000.`

Scientific Notation:

- `4.2e7`
- `-4.2E-7`

Floating-Point Numbers

The `float()` Function:

- `float()` → `0.0`
- `float("42")` → `42.0`
- `float(42)` → `42.0`
- `float(4.2)` → `4.2`

Floating-Point Numbers

The `float()` Function:

- `float()` → `0.0`
- `float("42")` → `42.0`
- `float(42)` → `42.0`
- `float(4.2)` → `4.2`

Special Values:

- `float("-inf")` → -∞
- `float("inf")` → ∞
- `float("nan")` → Not a Number

Table of Contents

Table of Contents

Arithmetic Operators

Arithmetic Operators

Binary:

- Addition: `5 + 2`
- Subtraction: `5 - 2`

Arithmetic Operators

Binary:

- Addition: `5 + 2`
- Subtraction: `5 - 2`

Unary:

- Plus: `+2`
- Minus: `-2`

Arithmetic Operators

Binary:

- Addition: `5 + 2`
- Subtraction: `5 - 2`
- Multiplication: `5 * 2`

Unary:

- Plus: `+2`
- Minus: `-2`

Arithmetic Operators

Binary:

- Addition: `5 + 2`
- Subtraction: `5 - 2`
- Multiplication: `5 * 2`
- Exponentiation: `5 ** 2`

Unary:

- Plus: `+2`
- Minus: `-2`

Arithmetic Operators

Binary:

- Addition: `5 + 2`
- Subtraction: `5 - 2`
- Multiplication: `5 * 2`
- Exponentiation: `5 ** 2` $\Leftrightarrow 5^2$

Unary:

- Plus: `+2`
- Minus: `-2`

Arithmetic Operators

Binary:

- Addition: `5 + 2`
- Subtraction: `5 - 2`
- Multiplication: `5 * 2`
- Exponentiation: `5 ** 2` $\Leftrightarrow 5^2$
- Division: `5 / 2`

Unary:

- Plus: `+2`
- Minus: `-2`

Arithmetic Operators

Binary:

- Addition: `5 + 2`
- Subtraction: `5 - 2`
- Multiplication: `5 * 2`
- Exponentiation: `5 ** 2` $\Leftrightarrow 5^2$
- Division: `5 / 2` $\Rightarrow 2.5$

Unary:

- Plus: `+2`
- Minus: `-2`

Arithmetic Operators

Binary:

- Addition: `5 + 2`
- Subtraction: `5 - 2`
- Multiplication: `5 * 2`
- Exponentiation: `5 ** 2` $\Leftrightarrow 5^2$
- Division: `5 / 2` $\Rightarrow 2.5$
- Integer Division: `5 // 2` $\Rightarrow 2$

Unary:

- Plus: `+2`
- Minus: `-2`

Arithmetic Operators

Binary:

- Addition: `5 + 2`
- Subtraction: `5 - 2`
- Multiplication: `5 * 2`
- Exponentiation: `5 ** 2` $\Leftrightarrow 5^2$
- Division: `5 / 2` $\Rightarrow 2.5$
- Floor Division: `5 // 2` $\Rightarrow 2$

Unary:

- Plus: `+2`
- Minus: `-2`

Arithmetic Operators

Binary:

- Addition: `5 + 2`
- Subtraction: `5 - 2`
- Multiplication: `5 * 2`
- Exponentiation: `5 ** 2` $\Leftrightarrow 5^2$
- Division: `5 / 2` $\Rightarrow 2.5$
- Floor Division: `5 // 2` $\Rightarrow 2$
- Modulus: `5 % 2`

Unary:

- Plus: `+2`
- Minus: `-2`

Arithmetic Operators

Binary:

- Addition: `5 + 2`
- Subtraction: `5 - 2`
- Multiplication: `5 * 2`
- Exponentiation: `5 ** 2` $\Leftrightarrow 5^2$
- Division: `5 / 2` $\Rightarrow 2.5$
- Floor Division: `5 // 2` $\Rightarrow 2$
- Modulus: `5 % 2` $\Rightarrow 1$

Unary:

- Plus: `+2`
- Minus: `-2`

Arithmetic Operators

Operator	Description	Precedence
`**`	Exponentiation	⬆ Highest
`+x`, `-x`	Unary plus and minus	
`*`, `/`, `//`, `%`	Multiplication, division, floor division, modulus	
`+`, `-`	Addition and subtraction	⬇ Lowest

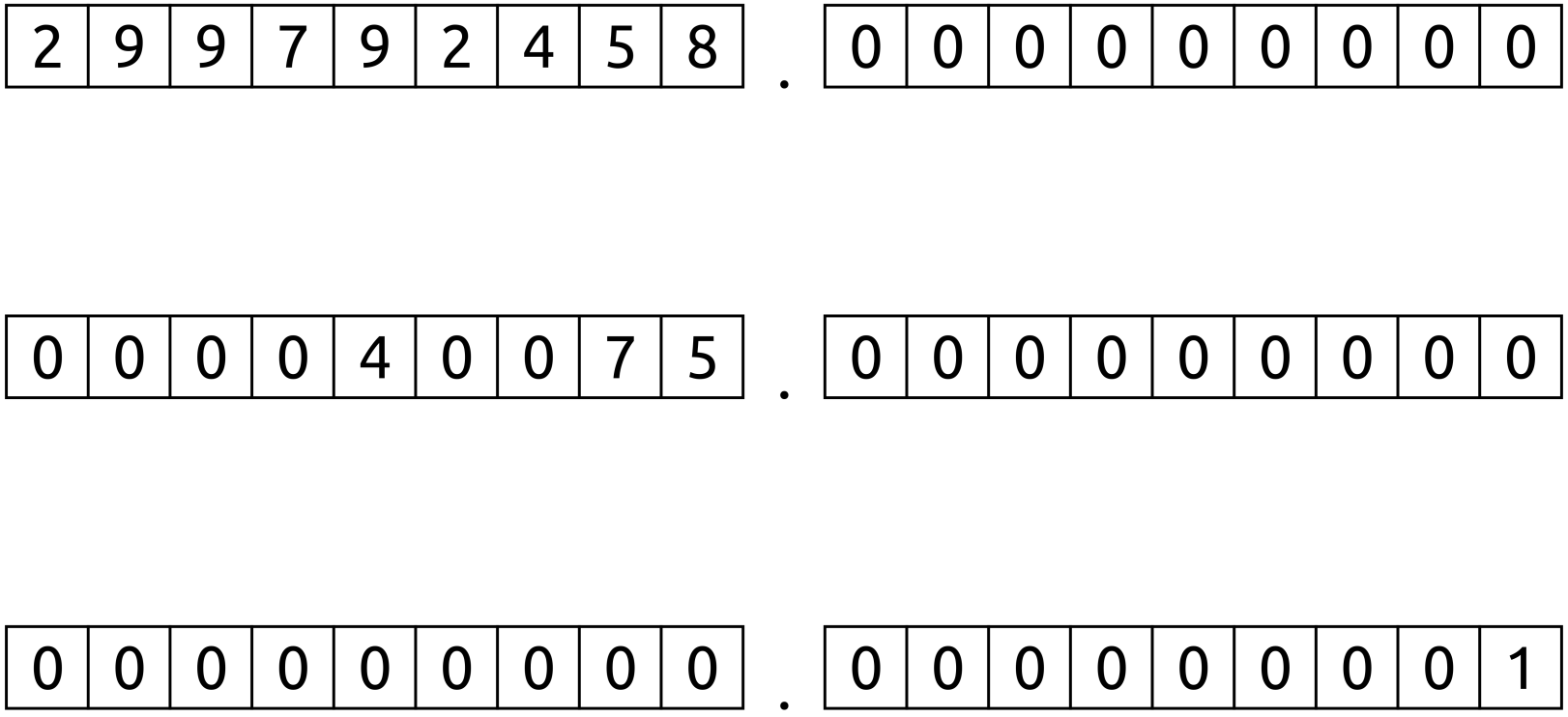
Table of Contents

Table of Contents

Floating-Point Representation Error

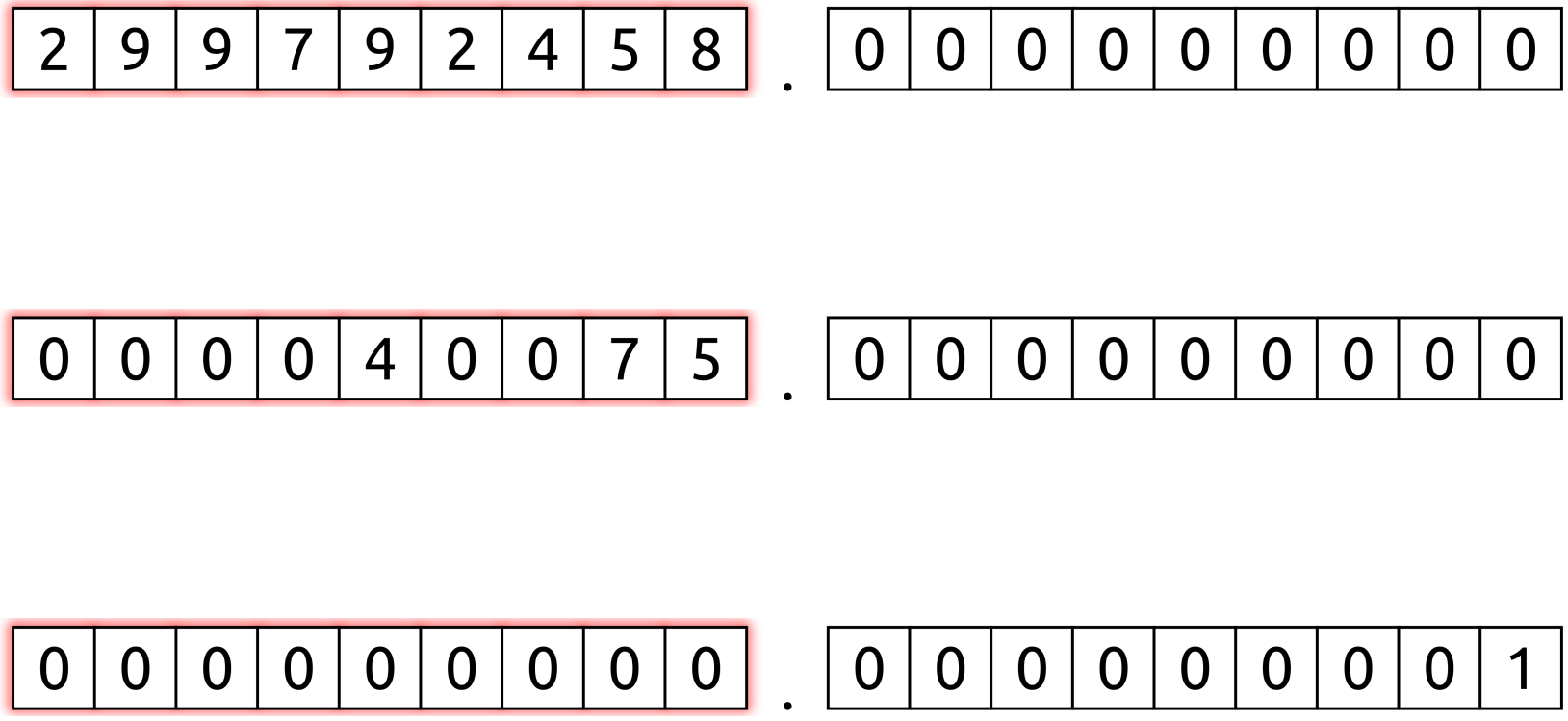
Floating-Point Representation Error

⚓ Fixed-Point Numbers



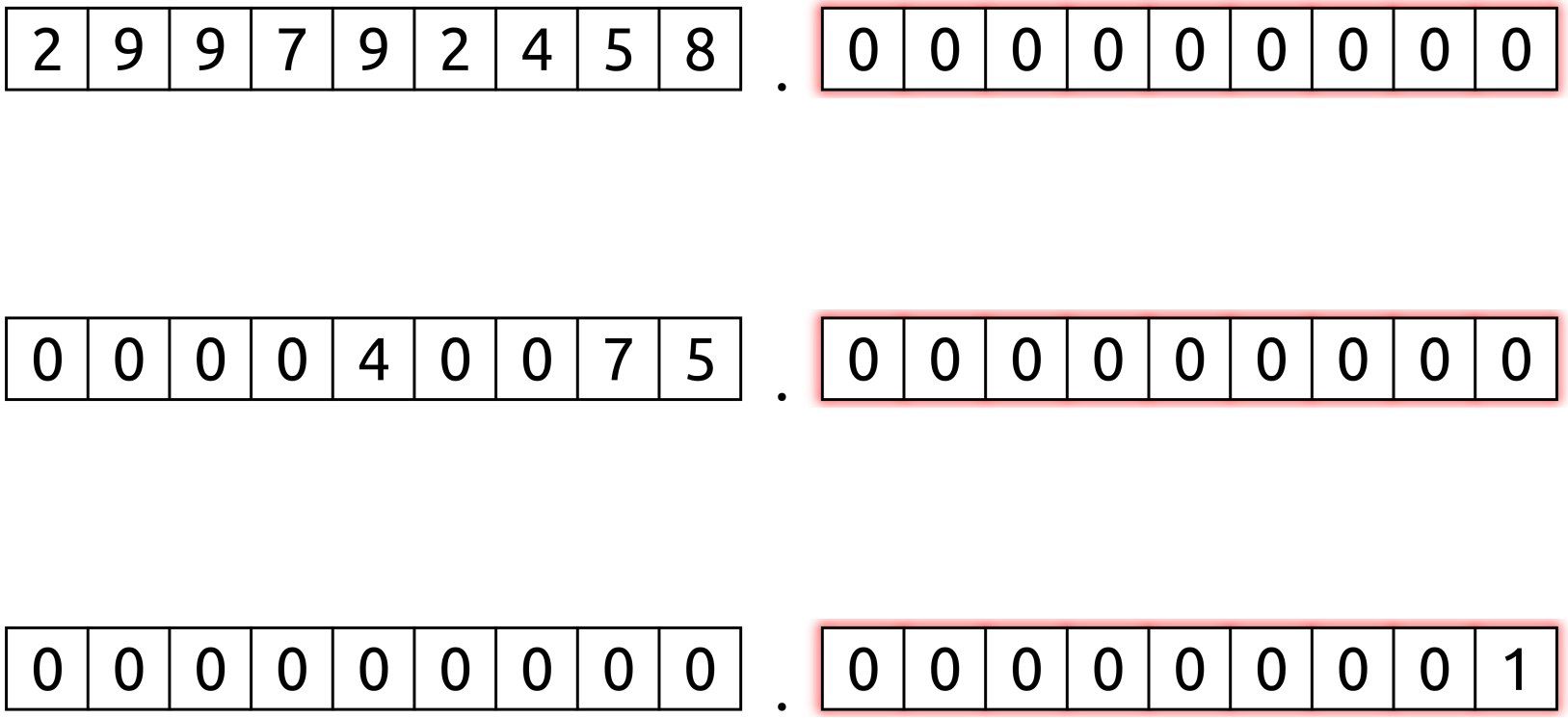
Floating-Point Representation Error

⚓ Fixed-Point Numbers



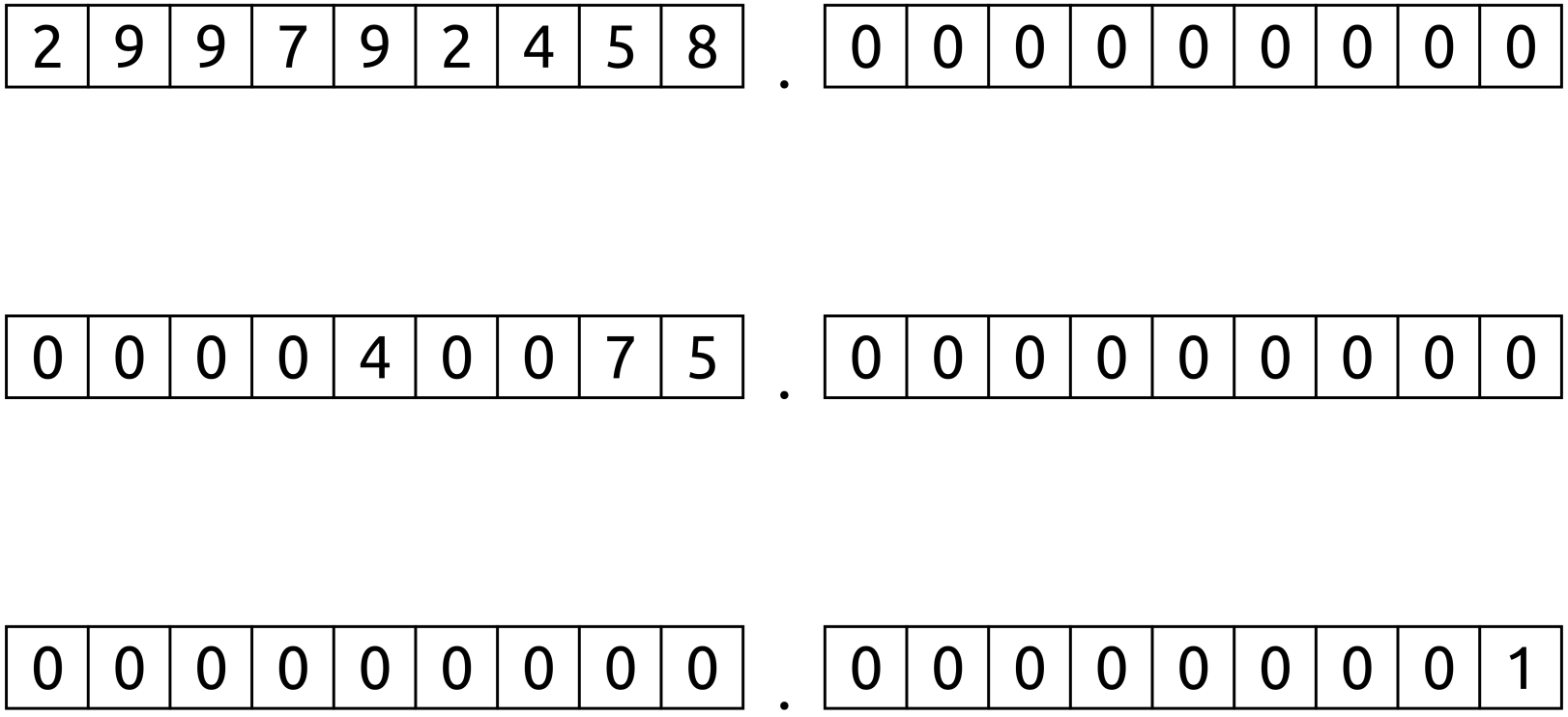
Floating-Point Representation Error

⚓ Fixed-Point Numbers



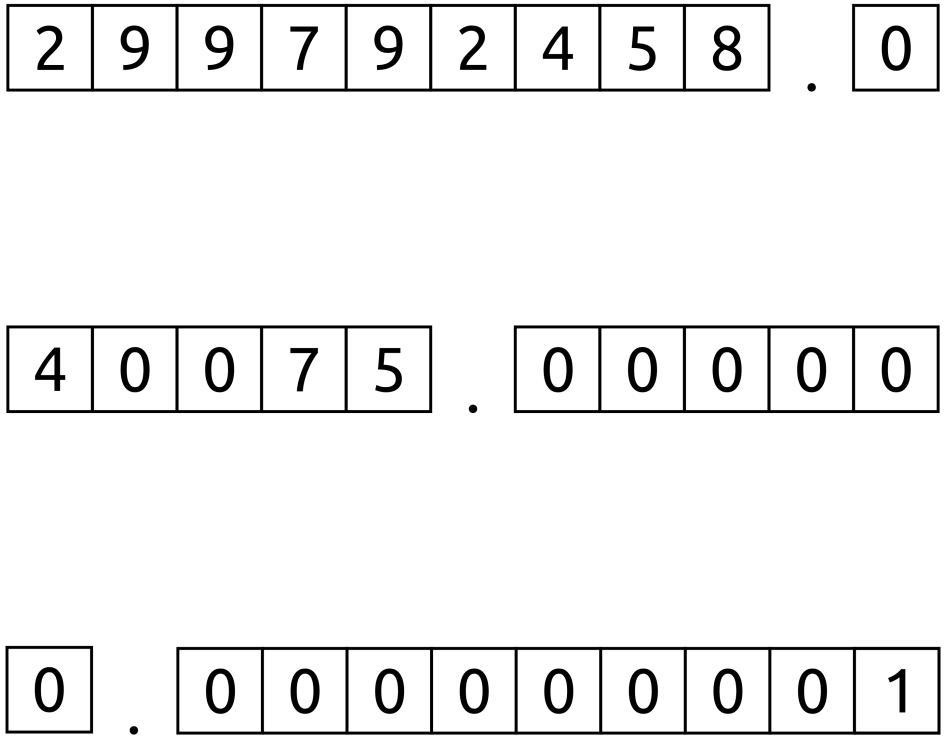
Floating-Point Representation Error

⚓ Fixed-Point Numbers



Floating-Point Representation Error

🚣 Floating-Point Numbers



Floating-Point Representation Error

🚣 Floating-Point Numbers

$$0.1 + 0.2 \neq 0.3$$

Table of Contents

Table of Contents

Math Functions and Number Methods

Math Functions and Number Methods

Math Functions:

- `round()`
- `abs()`
- `pow()`

Math Functions and Number Methods

Math Functions:

- `round()`
- `abs()`
- `pow()` ⇔ `**`

Math Functions and Number Methods

Math Functions:

- `round()`
- `abs()`
- `pow()` ⇔ `**`

Floating-Point Number Methods:

- `float.is_integer()`
- `float.as_integer_ratio()`

Math Functions and Number Methods

Math Functions:

- `round()`
- `abs()`
- `pow()` ⇔ `**`

Floating-Point Number Methods:

- `float.is_integer()`
- `float.as_integer_ratio()`

Integer Number Methods:

- `int.bit_length()`
- `int.bit_count()`

Table of Contents

Table of Contents

Formatting Numbers as Strings

Formatted String Literal	Sample Output
`f"{price:.2f}"`	“3.90”
`f"{price:15.2f}"`	“ 3.90”
`f"{price:<15.2f}"`	“3.90 ”
`f"{price:^15.2f}"`	“ 3.90 ”
`f"{price:015.2f}"`	“000000000003.90”
`f"{price:<015.2f}"`	“3.900000000000”
`f"{price:,.2f}"`	“1,999,999.99”

Formatting Numbers as Strings

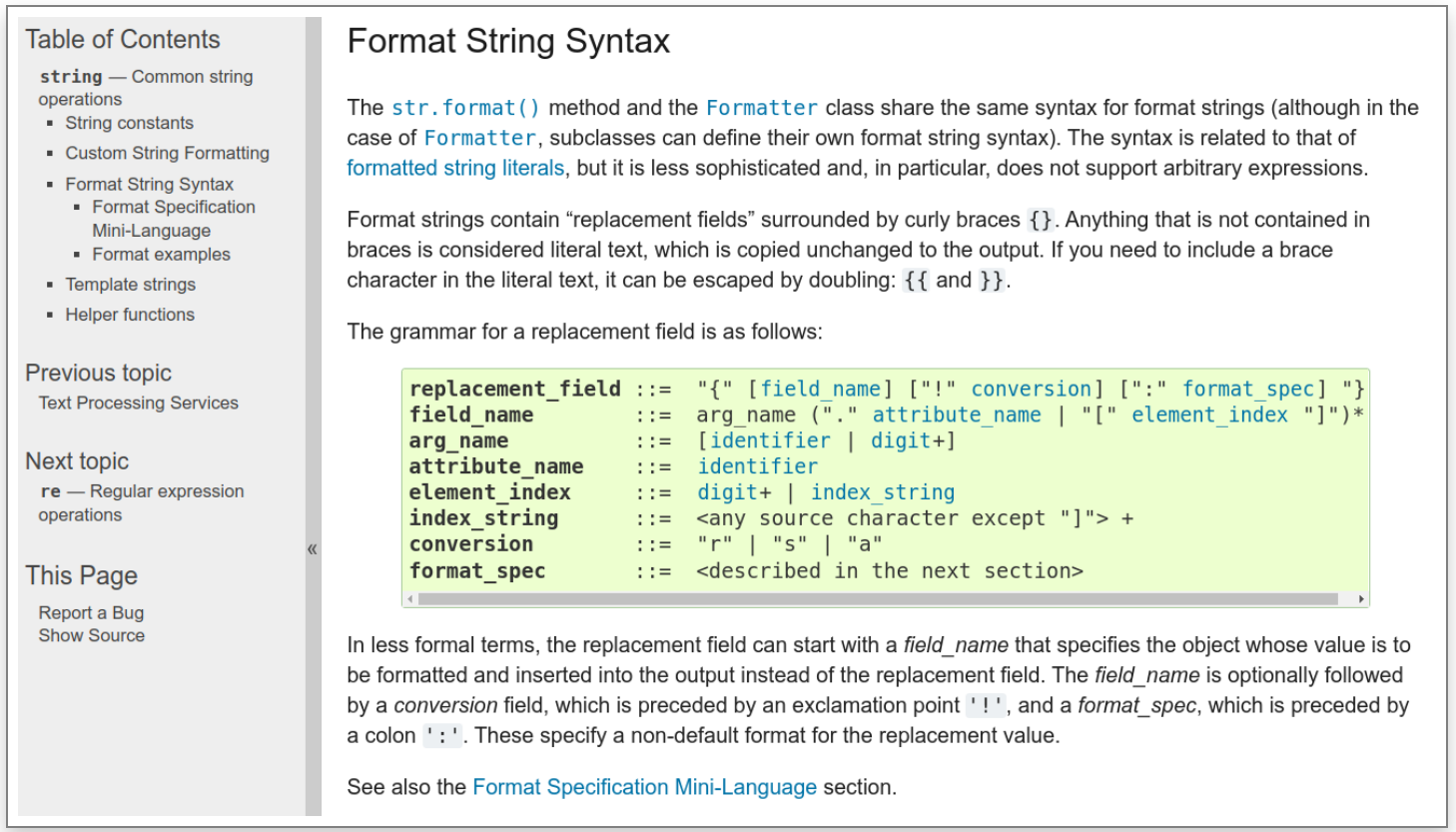


Table of Contents

- **string** — Common string operations
  - String constants
  - Custom String Formatting
  - Format String Syntax
    - Format Specification Mini-Language
    - Format examples
  - Template strings
  - Helper functions

Previous topic

Text Processing Services

Next topic

re — Regular expression operations

This Page

Report a Bug
Show Source

Format String Syntax

The `str.format()` method and the `Formatter` class share the same syntax for format strings (although in the case of `Formatter`, subclasses can define their own format string syntax). The syntax is related to that of formatted string literals, but it is less sophisticated and, in particular, does not support arbitrary expressions.

Format strings contain “replacement fields” surrounded by curly braces `{}`. Anything that is not contained in braces is considered literal text, which is copied unchanged to the output. If you need to include a brace character in the literal text, it can be escaped by doubling: `{{` and `}}`.

The grammar for a replacement field is as follows:

```
replacement_field ::=  "{" [field_name] ["!" conversion] [":" format_spec] "}"
field_name        ::=  arg_name ("." attribute_name | "[" element_index "]")*
arg_name          ::=  [identifier | digit+]
attribute_name    ::=  identifier
element_index     ::=  digit+ | index_string
index_string      ::=  <any source character except "]"> +
conversion        ::=  "r" | "s" | "a"
format_spec       ::=  <described in the next section>
```

In less formal terms, the replacement field can start with a *field_name* that specifies the object whose value is to be formatted and inserted into the output instead of the replacement field. The *field_name* is optionally followed by a *conversion* field, which is preceded by an exclamation point `'!'`, and a *format_spec*, which is preceded by a colon `':'`. These specify a non-default format for the replacement value.

See also the Format Specification Mini-Language section.

https://docs.python.org/3/library/string.html#format-string-syntax

Table of Contents

Table of Contents

Complex Numbers

Complex Numbers

Definition of a Complex Number:

- **Function:** `complex(3, 2)`
- **Literal:** `3 + 2j`

Complex Numbers

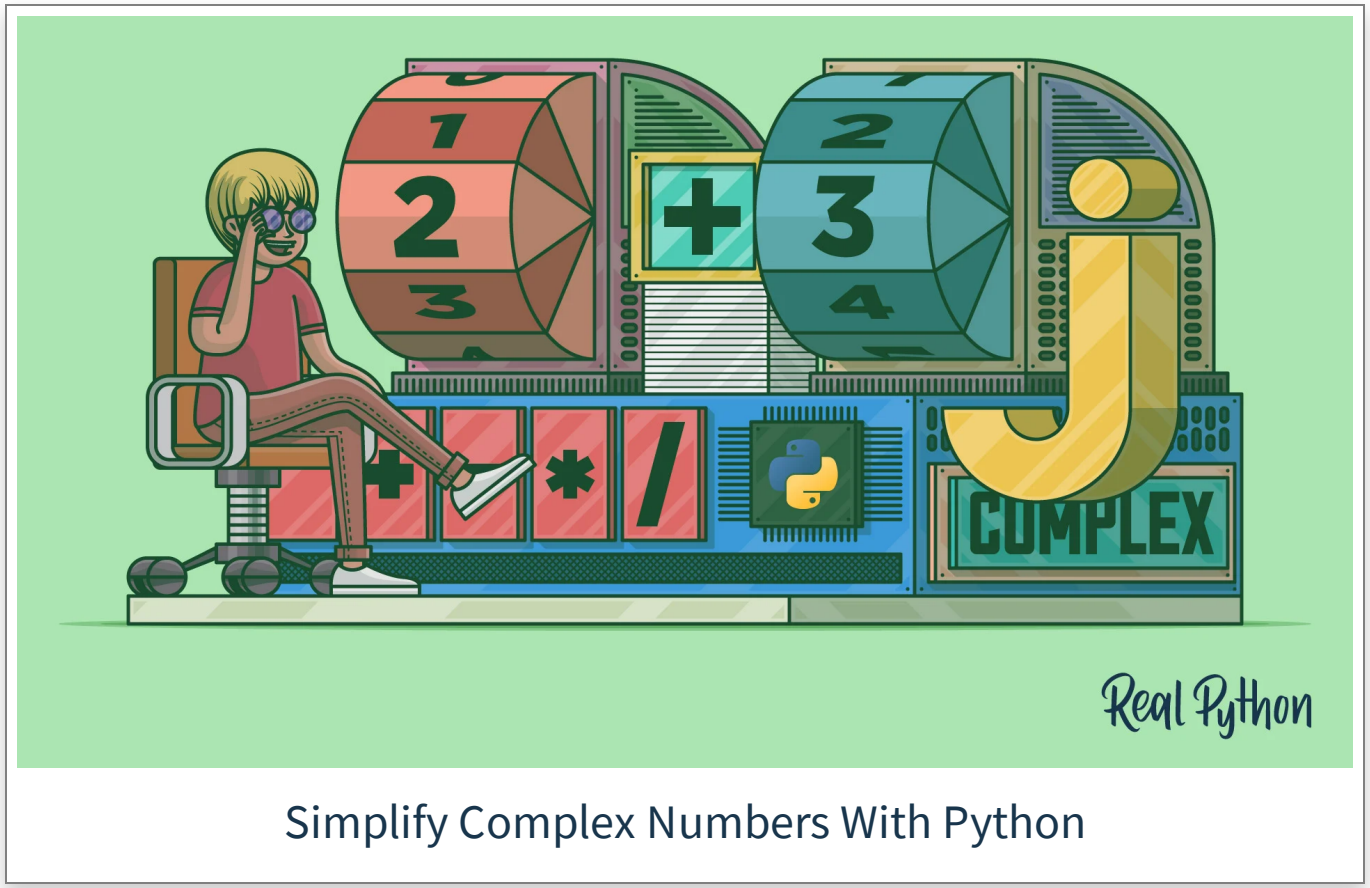
Definition of a Complex Number:

- **Function:** `complex(3, 2)`
- **Literal:** `3 + 2j`

Attributes and Methods:

- `z.real`
- `z.imag`
- `z.conjugate()`

Complex Numbers



Simplify Complex Numbers With Python

Table of Contents

Table of Contents

Summary

In this course you:

- Got an overview of the numeric types in Python
- Created integers, floats, and complex numbers
- Performed arithmetic operations
- Learned about the floating-point representation error
- Worked with math functions and number methods
- Formatted and displayed numbers as strings

Thank you!


PYTHON BASICS
Real Python
PYTHON BASICS
RP
1 2 3
4 5 6
7 8 9
. 0 =
+ - * /
// ** %
round( )
abs( )
pow( )
int( )
flt( )
j