# The Basics

## Values

42

12.345

“Hello!”

## Types

**int**eger

**float** (real number)

**str**ing (of characters)

## Expressions

34 * (23 + 14)

1.0 / 3.0

"Hel" + "lo!"

# Python and Expressions

- An expression **represents** something
  - Python *evaluates it* (turns it into a value)
  - Similar to what a calculator does
- Examples:
  - 2.3

    Literal
    (evaluates to self)

  - (3 * 7 + 2) * 0.1

    An expression with four
    literals and some operators

# Representing Values

- **Everything** on a computer reduces to numbers
  - Letters represented by numbers (ASCII codes)
  - Pixel colors are three numbers (red, blue, green)
  - So how can Python tell all these numbers apart?

**Memorize this definition!**

- **Type:**
  **A set of values and the operations on them.**
  - Examples of operations: +, -, /, *
  - The meaning of these depends on the type

# Example: Type **int**

- Type **int** represents integers
  - values: …, –3, –2, –1, 0, 1, 2, 3, 4, 5, …
    - Integer literals look like this: `1`, `45`, `43028030` (no commas or periods)
  - operations: `+`, `–`, `*`, `/`, `**`, unary `–`

    multiply

    to power of
- **Principle**: operations on **int** values must yield an **int**
  - **Example:** `1 / 2` rounds result down to 0
    - Companion operation: `%` (remainder)
    - `7 % 3` evaluates to 1, remainder when dividing 7 by 3
  - Operator / is not an **int** operation in Python 3 (use // instead)

# Example: Type float

- Type **float** (floating point) represents real numbers
  - values: distinguished from integers by decimal points
    - In Python a number with a "." is a **float** literal (e.g. `2.0`)
    - Without a decimal a number is an **int** literal (e.g. `2`)
  - operations: `+`, `–`, `*`, `/`, `**`, unary `–`
    - The meaning for floats differs from that for ints
    - **Example**: `1.0/2.0` evaluates to 0.5
- **Exponent notation** is useful for large (or small) values
  - `-22.51e6` is $-22.51 * 10^{6}$ or $-22510000$
  - `22.51e-6` is $22.51 * 10^{-6}$ or $0.00002251$

A second kind of **float** literal

# Floats Have Finite Precision

- Python stores floats as **binary fractions**
  - Integer mantissa times a power of 2
  - Example: 1.25 is $\underbrace{5}_{\text{mantissa}} * 2^{\underbrace{-2}_{\text{exponent}}}$
- Impossible to write most real numbers this way exactly
  - Similar to problem of writing 1/3 with decimals
  - Python chooses the closest binary fraction it can
- This approximation results in **representation error**
  - When combined in expressions, the error can get worse
  - **Example**: type `0.1 + 0.2` at the prompt >>>

# Example: Type bool

- Type boolean or **bool** represents logical statements
  - values: **True**, **False**
    - Boolean literals are just `True` and `False` (have to be capitalized)
  - operations: not, and, or
    - not b: **True** if b is false and **False** if b is true
    - b and c: **True** if both b and c are true; **False** otherwise
    - b or c: **True** if b is true or c is true; **False** otherwise

- Often come from comparing **int** or **float** values
  - Order comparison: i < j i <= j i >= j i > j
  - Equality, inequality: i == j i != j

"=" means something else!

# Example: Type **str**

- Type String or **str** represents text
  - values: any sequence of characters
  - operation(s): + (catenation, or concatenation)
- **String literal**: sequence of characters in quotes
  - Double quotes: `" abcex3$g<&"` or `"Hello World!"`
  - Single quotes: `'Hello World!'`
- Concatenation can only apply to strings.
  - `'ab' + 'cd'` evaluates to `'abcd'`
  - `'ab' + 2` produces an **error**

# Converting Values Between Types

- Basic form: *type*(*value*)
  - `float(2)` converts value 2 to type **float** (value now 2.0)
  - `int(2.6)` converts value 2.6 to type **int** (value now 2)
  - Explicit conversion is also called "casting"
- Narrow to wide: **bool** ⇒ **int** ⇒ **float**
  - *Widening*. Python does automatically if needed
    - **Example**: `1/2.0` evaluates to 0.5 (casts 1 to **float**)
  - *Narrowing*. Python *never* does this automatically
    - Narrowing conversions cause information to be lost
    - **Example**: `float(int(2.6))` evaluates to 2.0