COMP110

# CL02: Expressions

# Announcements

- Office Hours available Monday–Friday this week (11am-5pm)
- EX00 – *Hello, World!* – due Wednesday at 11:59pm
- Today: Paper + pencil / tablet + pencil

# Last Lecture

- Data Types
  - **`float`** (decimal, e.g. **`2.0`**)
  - **`int`** (whole number, e.g. **`2`**)
  - **`str`** (string of characters, e.g. **`“Hello”`**)
  - **`bool`** (evaluates to True or False, e.g, **`True`**)
- Check type
  - **`type()`**
- Change type
  - **`str(), float(), int()`**

# Review from Friday: Data Types

**Discuss these questions with your neighbor and jot the answers down.**

1. What is the difference between `int` and `float`?

2. Is there a difference between the following? What *type* of **literal** is each an example of?
    a. `“True”`
    b. `True`
    c. `TRUE`

3. What role do **types** play for data in Python?

# Review from Friday: `str` is a *Sequence* Type

**Discuss these questions with your neighbor and jot the answers down.**

1. What does the `len()` function evaluate to when applied to a `str` value? What will the expression `len(“cold”)` evaluate to?

2. Is there a difference between `“True”` and `‘True’`? What *type* of **literal** is each an example of?

3. What are the **square brackets** called in the following *expression*? What does the following expression evaluate to? `“The Bear”[4]`

4. Can a string be a number in Python? Explain.

# Expressions

- Fundamental building block in programs
- 2 main ideas behind expressions:
  - An expression *evaluates* to a *typed* value at runtime
  - An object’s *type* tells you what you can do with it

    An *expression* is an intent to do something
- Computer evaluates each expression in your program one step at a time
- Examples
  - 1 + 2 * 3
  - 1
  - 1.0 * 2.0
  - “Hello” + “ World!”
  - 1 > 3

# Numerical Operators

| Symbol | Operator Name | Example |
|---|---|---|
| `**` | Exponentiation | `2 ** 8` equivalent to $2^8$ |
| `*` | Multiplication | `10 * 3` |
| `/` | Division | `7 / 5` result is `1.4` |
| `//` | Integer Division | `7 // 5` result is `1` |
| `%` | Remainder “modulo” | `7 % 5` result is `2` |
| `+` | Addition | `1 + 1` |
| `-` | Subtraction | `111 - 1` |
| `-` | Negation | `-(1 + 1)` result is `-2` |

# Order Of Operations

- P ()
- E **
- MD * / %
- AS + -
- Tie? Evaluate *Left to Right*

# Addition +

- If numerical objects, add the values together
  - 1 + 1 “evaluates to” 2
  - 1.0 + 2.0 ⟶ 3.0
  - 1 + 2.0 ⟶ 3.0
- If strings, concatenate them
  - “Comp” + “110” ⟶ “Comp110”
- The result type depends on the operands
  - float + float ⟶ float
  - int + int ⟶ int
  - float + int ⟶ float
  - int + float ⟶ float
  - str + str ⟶ str

# Addition +

- If numerical objects, add the values together
  - 1 + 1 → 2
  - 1.0 + 2.0 → 3.0
  - 1 + 2.0 → 3.0
- If strings, concatenate them
  - “Comp” + “110” → “Comp110”
- The result type depends on the operands
  - float + float → float
  - int + int → int
  - float + int → float
  - int + float → float
  - str + str → str

**Question: What happens when you try to add incompatible types?**

# Subtraction/Negation -

- Meant strictly for numerical types
  - 3 - 2 → 1
  - 4.0 - 2.0 → 2.0
  - 4.0 - 2 → 2.0
  - - (1 + 1) → -2
- The result type depends on the operands
  - float - float → float
  - int - int → int
  - float - int → float
  - int - float → float

# Multiplication *

- If numerical objects, multiply the values
  - 1 * 1 → 1
  - 1.0 * 2.0 → 2.0
  - 1.0 * 2 → 2.0
- If string and int, repeat the string int’s number of times
  - “Hello” * 3 → “HelloHelloHello”
- The result type depends on the operands
  - float * float → float
  - int * int → int
  - float * int → float
  - int * float → float
  - str * int → str

**Question: What happens when you try `str * float`?**

# Division /

- Meant strictly for numerical types
    - 3 / 2 → 1.5
    - 4.0 / 2.0 → 2.0
    - 4 / 2 → 2.0
- Division results in a float
    - float / float → float
    - int / int → float
    - float / int → float
    - int / float → float
- For integer division // , the result type depends on the operands
    - int // int → int
    - float // float → float
    - float // int → float
    - int // float → float

# Remainder “modulo”

- Calculates the *remainder* when you divide two numbers
- Meant strictly for numerical types
  - 5 % 2 → 1
  - 6 % 3 → 0
- The result type depends on the operands
  - int % int → int
  - float % float → float
  - float % int → float
  - int % float → float
- Note:
  - If x is even, x % 2 → 0
  - If x is odd, x % 2 → 1

# Exponentiation **

- Meant strictly for numerical types
  - 2 ** 2 → 4
  - 2.0 ** 2.0 → 4.0
- The result type depends on the operands
  - float ** float → float
  - int ** int → int
  - float ** int → float
  - int ** float → float

## Order Of Operations

- P ()
- E **
- MD * / %
- AS + -
- Tie? Evaluate *Left to Right*

# Relational Operators

- Always result in a bool (True or False) value
- Equals (==) and Not Equal (!=)
  - ! is commonly used in programming languages to represent the word “not”
  - Can be used for all primitive types we’ve learned so far! (bool, int, float, str)
- Greater than (>), at least (>=), less than (<), at most (<=)
  - Just use on floats and ints
  - (Can *technically* use on all primitive types, but it might not evaluate in ways you’d expect!)

# Relational Operators

| Operator Symbol | Verbalization | True Ex. | False Ex. |
|---|---|---|---|
| == | Is equal to? | 1 == 1 | 1 == 2 |
| != | Is NOT equal to? | 1 != 2 | 1 != 1 |
| > | Is greater than? | 1 > 0 | 0 > 1 |
| >= | Is at least? | 1 >= 0 or 1 >= 1 | 0 >= 1 |
| < | Is less than? | 0 < 1 | 1 < 0 |
| <= | Is at most? | 0 <= 1 or 1 <= 1 | 1 <= 0 |

# Practice: Operators and Expressions

**Discuss these questions with your neighbor and jot the answers down.**

1. What is the result of evaluating `10 % 3`? What about `10 // 3`? **10 ** 3**?

2. Is there an error in the expression, `“CAMP” + 110`? If so, how would you fix it such that the `+` symbol is evaluated to be **concatenation**?

3. What is the evaluation of the expression `10 / 4`? What types are the operands (`10` and `4`), what type does the expression evaluate to?

4. What is the evaluation of the expression `2 - 6 / 3 + 4 * 5`?

# Practice! Simplify and Type

- 2  + 4 / 2 * 2
- 220 >= int((“1” + “1” + “0”) * 2)

# Simplify: 2 + 4 / 2 * 2

(Reminder: P E M D A S)

Simplify: 2 + 4 / 2 * 2

# What type is 2 + 4 / 2 * 2?

# Simplify:

```
220 >= int((“1” + “1” + “0”) * 2)
```

## Mods Practice! Simplify

- 7 % 2
- 8 % 4
- 7 % 4
- Any even number % 2
- Any odd number % 2