# COMP 110

# CL02 - Expressions

# 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, 2 >= 3)
- Check type
  - type()
- Change type
  - str(), float(), int()

# Expressions

- Something that *evaluates* at runtime
- Every expression evaluates to a specific typed value
- Examples
  - 1 + 2 * 3
  - 1
  - 1.0 * 2.0
  - "Hello" + " World!"
  - 1 > 3

# Numerical Operators

| Operator Name | Symbol |
|---|---|
| Addition | + |
| Subtraction/Negation | - |
| Multiplication | * |
| Division | / |
| Exponentiation | ** |
| Remainder "modulo" | % |

# Addition +

- If numerical objects, add the values together
  - 1 + 1 → 2
  - 1.0 + 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
- 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
- If string and int, repeat the string
  - "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

# 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

# Exponentiation **

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

## Order Of Operations

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

# Relational Operators

| Operator Name | Symbol |
| --- | --- |
| Equal? | == |
| Less than? | < |
| Greater than? | > |
| Less than or equal to? (At most) | <= |
| Greater than or equal to? (At least) | >= |
| Not equal? | != |

# Relational Operators

- Always result in a bool (True or False)
- Equals (==) and Not Equal (!=)
    - Can be used for all primitive types we've learned so far! (bool, int, float, str)
- Every other type
    - Just use on floats and ints
    - (Can *technically* use on all primitive types)

# 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