Helping You Succeed in this Class

- **Consultants.** ACCEL Lab Green Room
  - Daily office hours (see website) with consultants
  - Very useful when working on assignments
- **AEW Workshops**. Additional discussion course
  - Runs parallel to this class – completely optional
  - See website; talk to advisors in Olin 167.
- **Piazza.** Online forum to ask and answer questions
  - Go here first **before** sending question in e-mail
- **Office Hours.** Talk to the professor!
  - Available in Bailey basement between lectures

iClickers

- Have you registered your iclicker?
- If not, visit (now with no surcharge!)
  - https://cs1110.cs.cornell.edu/py/clicker
- See the course web page for more:
  - http://www.cs.cornell.edu/courses/cs1110/2019fa
  - Click "Materials/Textbook"
  - Look under "iClickers"

Converting Values Between Types

- Basic form: *type*(*expression*)
  - This is an expression
  - Evaluates to value, converted to new type
  - This is sometimes called **casting**
- **Examples:**
  - float(2) evaluates to 2.0 (a **float**)
  - int(2.6) evaluates to 2 (an **int**)
  - Note information loss in 2nd example

Converting Values Between Types

- Conversion is measured *narrow* to *wide*

bool ⇒ int ⇒ float

- **Widening:** Convert to a wider type
  - Python does automatically
  - **Example:** 1/2.0 evaluates to 0.5
- **Narrowing:** Convert to a narrower type
  - Python never does automatically
  - **Example:** float(int(2.6)) evaluates to 2.0

Operator Precedence

- What is the difference between these two?
  - 2*(1+3) **add, then multiply**
  - 2*1 + 3 **multiply, then add**
- Operations are performed in a **set order**
  - Parentheses make the order explicit
  - What happens when no parentheses?
- **Operator Precedence**: The *fixed* order Python processes operators in *absence* of parentheses

Precedence of Python Operators

- **Exponentiation**: **
- **Unary operators**: + −
- **Binary arithmetic**: * / %
- **Binary arithmetic**: + −
- **Comparisons**: < > <= >=
- **Equality relations**: == !=
- **Logical not**
- **Logical and**
- **Logical or**

- Precedence goes downwards
  - Parentheses highest
  - Logical ops lowest
- Same line = same precedence
  - Read "ties" left to right
  - Example: 1/2*3 is (1/2)*3

- Section 2.5 in your text
- See website for more info
- Was major portion of Lab 1

Variables

- A **variable**
  - is a **box** (memory location)
  - with a **name**
  - and a **value** in the box
- Examples:

x | 5 | Variable **x**, with value 5 (of type **int**)

area | 20.1 | Variable **area**, w/ value 20.1 (of type **float**)

Using Variables

- Variables can be used in expressions
  - Evaluate to the value that is in the box
  - **Example**: x | 5 | **1 + x** evaluates to **6**
- Variables can change values
  - **Example**: x | ~~5~~ 1.5 | **1 + x** evaluates to **2.5**
  - Can even change the **type** of their value
  - Different from other languages (e.g. Java)

Variables and Assignment Statements

- Variables are created by **assignment statements**

```
x = 5
```

the value

the variable

x | 5

- This is a **statement**, not an **expression**
  - **Expression**: Something Python turns into a value
  - **Statement**: Command for Python to do something
  - Difference is that has no value itself
- **Example**:

```
>>> x = 5
(NOTHING)
```

But can now use x as an expression

Assignments May Contain Expressions

- **Example**: `x = 1 + 2`
  - Left of equals must always be variable: ~~`1 + 2 = x`~~
  - Read assignment statements right-to-left!
  - Evaluate the expression on the right
  - Store the result in the variable on the left
- We can include variables in this expression
  - **Example**: `x = y+2`
  - **Example**: `x = x+2`

x | 5

y | 2

This is not circular! Read right-to-left.

Dynamic Typing

- Python is a **dynamically typed language**
  - Variables can hold values of any type
  - Variables can hold different types at different times
- The following is acceptable in Python:

```
>>> x = 1
>>> x = x / 2.0
```

← x contains an **int** value

← x now contains a **float** value

- Alternative is a **statically typed language**
  - Each variable restricted to values of just one type
  - This is true in Java, C, C++, etc.

Dynamic Typing

- Often want to track the type in a variable
  - What is the result of evaluating x / y?
  - Depends on whether x, y are **int** or **float** values
- Use expression `type(<expression>)` to get type
  - `type(2)` evaluates to `<type 'int'>`
  - `type(x)` evaluates to type of contents of x
- Can use in a boolean expression to test type
  - `type('abc') == str` evaluates to **True**