Lecture 1

Course Overview, Python Basics

CS 1133 Spring 2018: Craig Frey

- **Outcomes:**
  - **Competency** with basic Python programming
    - Ability to create Python modules and programs
    - Ability to use the most common built-in data types
  - **Knowledge** of object-oriented programming
    - Ability to recognize and use objects in Python.
    - Ability to understand classes written by others.
- **Website:**
  - www.cs.cornell.edu/courses/cs1133/2018sp/

About Your Instructor

- **Teaches**
  - **CS 2024 C++ Programming**
  - **CS 2049 Int iPhone Development**
  - **CS 1130 Transition to Object Oriented Programming**
- **Developer for facilities**
  - **Utilities billing / $5 million / month**

Class Structure

- **Lectures.** Every Monday/Friday
  - Similar to lectures in CS 1110
  - Some interactive demos; bring laptops
- **Labs.** Every Wednesday
  - Self-guided activities to give practice
  - Several instructors on hand to help out
- **Consulting Hours**: 4:30-9:30, Sunday-Thursday
  - Open office hours with (CS 1110) staff
  - Open to CS 1133 students as well
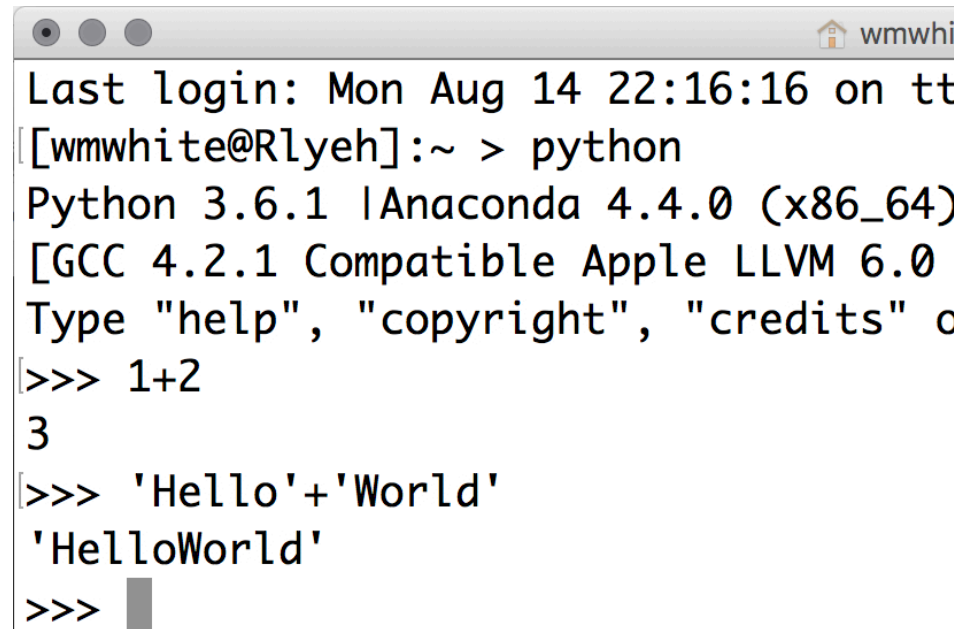 - Held in ACCEL Labs, Carpenter Hall

Grading Policy

- There will be two assignments
  - Course is not long enough to do much more
  - But both will involve programming
- Must earn 85% to pass an assignment
  - Get two more attempts if you fail
  - But you must meet the posted deadlines!
- Must pass both assignments
- No exams; labs are not graded

Getting Started with Python

- Designed to be used from the "command line"
  - OS X/Linux: **Terminal**
  - Windows: **Command Prompt**
  - Purpose of the first lab

- Once installed type "python"
  - Starts an *interactive shell*
  - Type commands at >>>
  - Shell responds to commands

- Can use it like a calculator
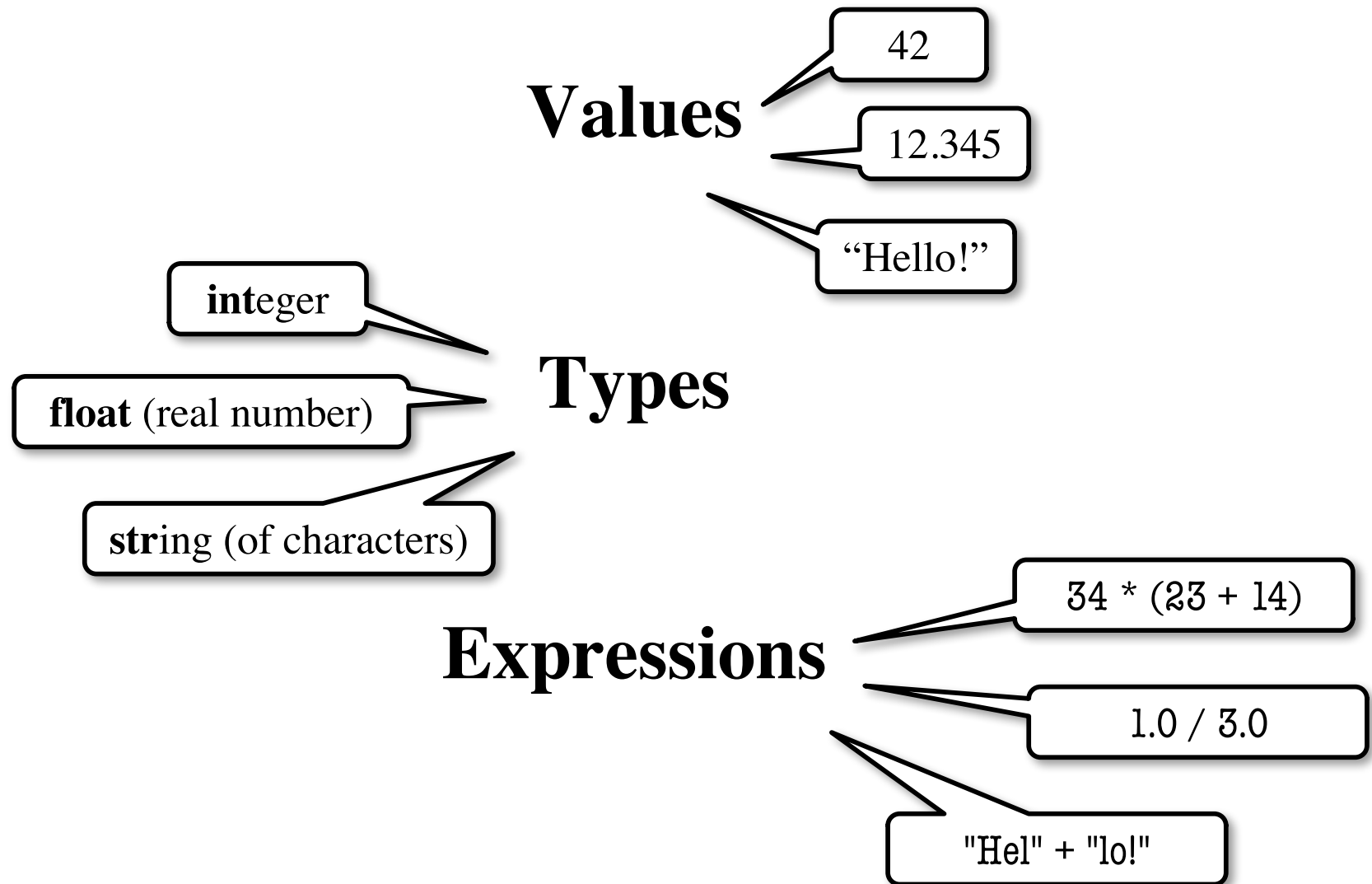 - Use to evaluate *expressions*

```
Last login: Mon Aug 14 22:16:16 on tt
[[wmwhite@Rlyeh]:~ > python
Python 3.6.1 |Anaconda 4.4.0 (x86_64)
[GCC 4.2.1 Compatible Apple LLVM 6.0
Type "help", "copyright", "credits" c
[>>> 1+2
3
[>>> 'Hello'+'World'
'HelloWorld'
>>>
```

This class uses Python 3.6

The Basics

Values

42

12.345

"Hello!"

integer

float (real number)

Types

string (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

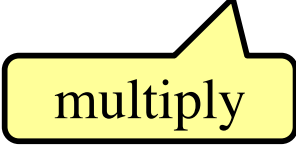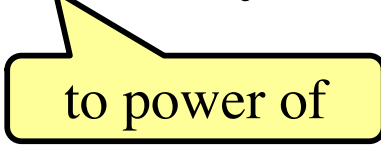   An expression with four
    literals and some operators

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?

- **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

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 -
    - Notice that float has a different division operator
    - **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

Representation Error

- Python stores floats as **binary fractions**
  - Integer mantissa times a power of 2
  - Example: 12.5 is   100 * 2^{-3}

  mantissa   **exponent**

- Impossible to write every number 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**

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'`

      > The meaning of + depends on the **type**
    - `'ab' + 2` produces an **error**

Summary of Basic Types

- Type **int**:
  - **Values**: integers
  - **Ops**: +, −, *, //, %, **
- Type **float**:
  - **Values**: real numbers
  - **Ops**: +, −, *, /, **
- Type **bool**:
  - **Values**: **True** and **False**
  - **Ops**: not, and, or

- Type **str**:
  - **Values**: string literals
    - Double quotes: "abc"
    - Single quotes: 'abc'
  - **Ops**: + (concatenation)

Will see more types
in the next week

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** $\Rightarrow$ **int** $\Rightarrow$ **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

Operator Precedence

- What is the difference between the following?
  - ■ 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 there are 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 2.7 in your text

- See website for more info

- Major portion of Lab 1

Expressions vs Statements

Expression

- **Represents** something
  - Python *evaluates it*
  - End result is a value
- Examples:
  - 2.3 ← Literal
  - (3+5)/4 ← Complex Expression

Statement

- **Does** something
  - Python *executes it*
  - Need not result in a value
- Examples:
  - print('Hello')
  - import sys

Will see later this is not a clear cut separation

Variables (Section 2.1)

- A **variable** is
  - a **named** memory location (**box**),
  - 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**)

- Variable names must start with a letter
  - So 1e2 is a **float**, but e2 is a variable name

Variables and Assignment Statements

- Variables are created by **assignment statements**
  - Create a new variable name and give it a value

    x = 3   **the value** / **the variable**

- This is a **statement**, not an **expression**

  - Tells the computer to DO something (not give a value)
  - Typing it into >>> gets no response (but it is working)

- Assignment statements can have expressions in them
  - These expressions can even have variables in them
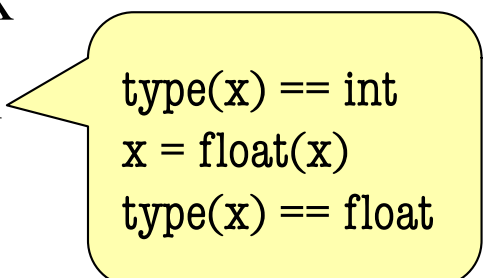
    x = x + 2   **the expression** / **the variable**

Dynamic Typing

- Python is a **dynamically typed language**
  - Variables can hold values of any type
  - Variables can hold different types at different times
  - Use `type(x)` to find out the type of the value in x
  - Use names of types for conversion, comparison

    > `type(x) == int`
    > `x = float(x)`
    > `type(x) == float`

- The following is acceptable in Python:

  `>>> x = 1` ← x contains an **int** value

  `>>> x = x / 2.0` ← x now contains a **float** value

- Alternative is a **statically typed language** (e.g. Java)
  - Each variable restricted to values of just one type

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**