CS 1133 Fall 2017: Walker White

- **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/cs1133/2017fa/

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!"

Types
- **int**eger
- **float** (real number)
- **str**ing (of characters)

Expressions
- 34 * (23 + 14)
- 1.0 / 3.0
- "Hel" + "lo!"

Type: Set of values and the operations on them

- 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

Type: Set of values and the operations on them

- 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

Type: Set of values and the operations on them

- Type boolean or **bool**:
  - 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 || 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!

Type: Set of values and the operations on them

- 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**

The meaning of + depends on the **type**

Expressions vs Statements

Expression	Statement
• **Represents** something	• **Does** something
▪ Python *evaluates it*	▪ Python *executes it*
▪ End result is a value	▪ Need not result in a value
• Examples:	• Examples:
▪ `2.3` (Literal)	▪ `print('Hello')`
▪ `(3+5)/4` (Complex Expression)	▪ `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 = x / 2.0
```

← x contains an **int** value

← x now contains a **float** value

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