Lecture 1

Types, Expressions, & Variables
About Your Instructor

• **Director**: GDIAC
  - Game Design Initiative at Cornell
  - Teach game design
• (and CS 1110 in fall)
Helping You Succeed in this Class

• **Consultants.** ACCEL Lab Green Room
  - Daily office hours (see website) with consultants
  - Very useful when working on assignments

• **Piazza.** Online forum to ask/answer questions
  - Go here first *before* sending question in e-mail

• **Office Hours.** Talk to the professor
  - Carpenter Hall Atrium on Tu Th 10-11 am
  - Otherwise, in 4118 Upson Hall
  - Open door policy (if door open, come in)
Office Hours this Semester

classroom

Office Hours Tue & Thu 10-11

9/3/13 Variables & Assignment
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*

This class uses Python 2.7.x
- Python 3 is too cutting edge
- Minimal software support
The Basics

Values

42
12.345
"Hello!"

Types

integer
float (real number)
string (of characters)

Expressions

34 * (23 + 14)
1.0 / 3.0
"Hel" + "lo!"
• An expression **represents** something
  ▪ Python *evaluates it* (turns it into a value)
  ▪ Similar to what a calculator does

• Examples:
  ▪ 2.3
  ▪ (3 * 7 + 2) * 0.1
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?

Memorize this definition!
Write it down several times.

• **Type:**
  A set of values and the operations on them.
  - Examples of operations: +, -, /, *
  - The meaning of these depends on the type
Type: Set of values and the operations on them

• Type **int** (integer):
  - **values**: …, –3, –2, –1, 0, 1, 2, 3, 4, 5, …
  - **operations**: +, –, *, /, **, unary –

  “Whole” numbers w/o decimals
  multiply
  to power of

• **Principal**: 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 (use // instead)
Type: Set of values and the operations on them

- **Type float** (floating point):
  - **values**: (approximations of) real numbers
    - 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 −
    - But meaning is different for floats
    - **Example**: 1.0/2.0 evaluates to 0.5

- **Exponent notation** is useful for large (or small) values
  - \(-22.51\text{e}6\) is \(-22.51 \times 10^6\) or \(-22510000\)
  - \(22.51\text{e}−6\) is \(22.51 \times 10^{-6}\) 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 $100 \times 2^{-3}$

• 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 >>>
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:**
  - values: any sequence of characters
  - operation(s): + (catenation, or concatenation)

- **String literal:** sequence of chars in quotes
  - Double quotes: " abc+x$g<!" or "Hello World!"
  - Single quotes: 'Hello World!'

- Concatenation can only apply to Strings.
  - "ab" + "cd" evaluates to "abcd"
  - "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 a few weeks
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 ⇒ int ⇒ 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
Casting: Converting Value Types

• Basic form: `type(value)`
  - `float(2)` casts value 2 to type `float` (value now 2.0)
  - `int(2.56)` casts value 2.56 to type `int` (value is now 2)

• Narrow to wide: `bool ⇒ int ⇒ float`
  - **Widening Cast.** Python does automatically if needed
    - Example: `1/2.0` evaluates to 0.5 (casts 1 to `float`)
  - **Narrowing Cast.** Python *never* does automatically
    - Narrowing casts cause information to be lost
    - Example: `float(int(2.56))` evaluates to 2.0
Expressions vs Statements

**Expression**
- **Represents** something
  - Python *evaluates it*
  - End result is a value

- **Examples:**
  - 2.3
  - (3+5)/4

**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

<table>
<thead>
<tr>
<th>x</th>
<th>5</th>
<th>Variable x, with value 5 (of type int)</th>
</tr>
</thead>
<tbody>
<tr>
<td>area</td>
<td>20.1</td>
<td>Variable area, w/ value 20.1 (of type float)</td>
</tr>
</tbody>
</table>

• 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
    \[ x = 3 \]
  ▪ 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
    \[ x = x + 2 \]
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
- The following is acceptable in Python:
  
  ```python
  >>> x = 1
  >>> x = x / 2.0
  >>> x
  0.5
  >>> type(x) == int
  False
  >>> x = float(x)
  >>> type(x) == float
  True
  ```
- 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`