main ( )

10/9/18

Journal

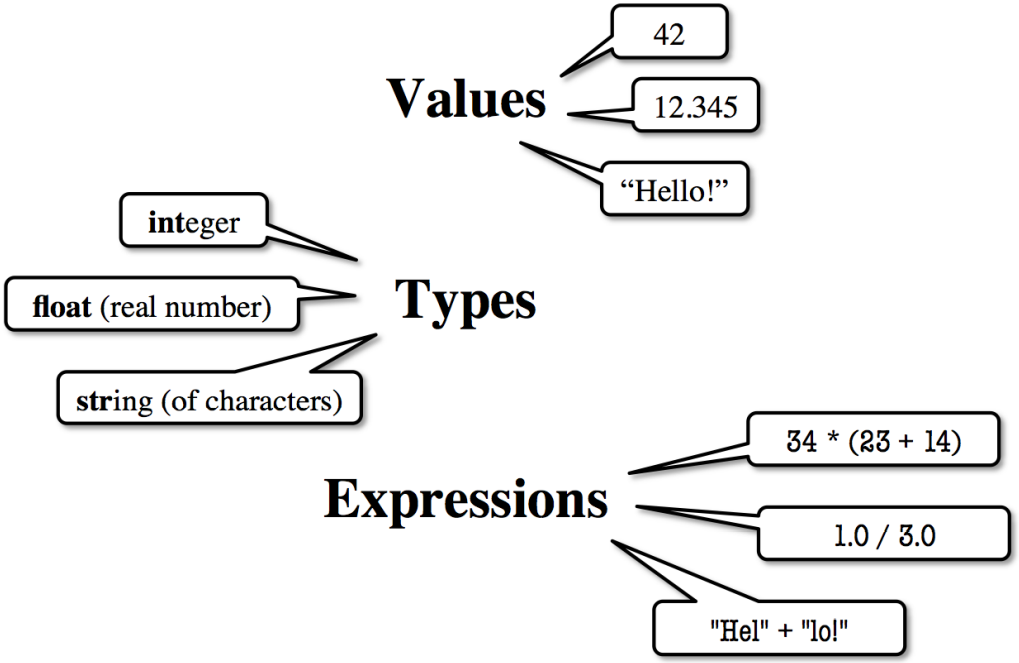
Spend 10 minutes working on the code from yesterday.
*Did you run into any errors?
*How did you feel once you fixed them?

Types and Evaluation

- Students will program four different types of values: floats, ints, strings, and booleans (briefly).
- Students will use expressions that are evaluated by Python and result in a value.
- Students will explain that statements are executed by Python and may not result in a value.
- Students will choose meaningful names for variables and procedures to help people better understand programs.
- Students will demonstrate that numbers and numerical concepts are fundamental to programming.
- Integers may be constrained in the maximum and minimum values that can be represented in a program because of storage limitations.
- Real numbers are approximated by floating-point representations that do not necessarily have infinite precision.
- Mathematical expressions using arithmetic operators are part of most programming languages.

CS Matters

The Basics



Representing Values

- **EVERYTHING** on a computer reduces to numbers
  - Letters represented by numbers (unicode)
  - Pixel colors are three numbers (red, blue, green)
  - So how can Python tell these numbers apart?

- **Type:**

A set of values and the operations on them.

  - Examples of operations: +, -, /, *
  - The meaning of these depends on the type

Expressions vs. Statements

Expression

- **Represents** something
  - Python *evaluates it*
  - End result is a value
- Examples:
  - 2.3 (Literal)
  - (3 * 7 + 2) * 0.1

An expression with four literals and some operators

Statement

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

Type: int

4/3 = 1
5%3 = 2

- Type **int** (integer):
  - 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

+ – * / ** asterisk power

- **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 (use // instead)

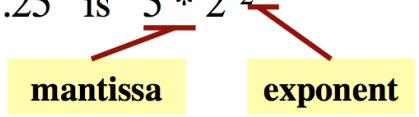
Type: float

- 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 −
    - The meaning for floats differs from that for ints
    - **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

Floats Have Finite Precision

- Python stores floats as **binary fractions**
  - Integer mantissa times a power of 2
  - Example: 1.25 is $5 * 2^{-2}$



- Impossible to write most real numbers 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: str → "1" + "2" = "12"

- Type String or **str**:
  - 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**

Computer + Science

ComputerScience

Type: bool

testing = =
assign =

not equal !=

not

"bye" or "BYE" or "Bye"

- 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 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!

Runestone Chapters on Variables and Expressions

Simple Python Data
Variables, Expressions and Statements (Video is 8:04)
Values and Data Types
Type conversion functions
Variables
Variable Names and Keywords
Statements and Expressions
Operators and Operands
Input
Order of Operations
Reassignment
Updating Variables

Make any notes for yourself in your journal.

Finish as homework if necessary.

Runestone: Values and Expressions