Variables and Values
Names

• Variables and functions both have names
  – Names must begin with a letter and may contain letters, digits, and underscores
  – Names are case-sensitive – total, TOTAL, and Total are three different names
  – There are a number of reserved words, such as if and while, that cannot be used as the name of a variable or function
Style

- **Style** is all the little things, not required by the language, that make a program easier or harder to read
  - For example: Lines of code should not extend past about column 80, because long lines are harder to read
- Some style rules are just commonly accepted conventions
- **Style rule:** Variable names should always begin with a lowercase letter (class/object names begin a capital letter)
- In Java, **camelCase** naming is the norm, e.g., `sumOfAngles`
- In Python, more common to use underscore, `sum_of_angles`
Importance of Style

• Programs are read more often than they are written
  – Estimates range from 20 times to 50 times
• The easier a program is to read, the easier it is to understand, debug, modify
• Style not so important for very small programs (<500 lines of code)
  – This class is not about writing very small programs!
• I’m not a style nazi
  – Won’t enforce camelCase vs. underscore but be consistent!
  – If you’re not consistent, people reading your code will be confused
Indentation not style in Python!

- In Python, indentation isn’t a question of style
  - It’s the law
  - In java, indentation doesn’t matter, so it is a question of style!

- Indentation determines where code blocks start and end

- Think control statements
  - if
  - while
Numbers

• In math, integers may be arbitrarily large
• Most programming languages have finite number precision
  – For example 64 bits, i.e., biggest number is \( \sim 2^{64} \)
• Python has infinite integer precision

• In math, real numbers also have infinite precision
  – Pi has been calculated to a few trillion digits
• In programming, floating point numbers have limited precision
  – E.g., 64 bits precision
  – \( 10/3 = 3.33333333333335 \)
Number Systems

• Can represent numbers **binary** (base 2), **octal** (base 8), **decimal** (base 10) or **hexadecimal** (base 16)
  – Humans almost exclusively use base 10
  – Computers use binary because of how they work (current, no current)
• The binary (base 2) number system uses two “binary digits,” (abbreviation: bits) -- 0 and 1
• The octal (base 8) number system uses eight digits: 0, 1, 2, 3, 4, 5, 6, 7
• The decimal (base 10) number system uses ten digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
• The hexadecimal, or “hex” (base 16) number system uses sixteen digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
• Regardless of how they are written, numbers are stored in binary and displayed in decimal
Numbers as viewed by the computer

• Integers are simply converted from binary to decimal
  – E.g., 26 = 11010
  – $2 \times 10^1 + 6 \times 10^0 = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$
  – Usually 32 or 64 bits allocated per integer (infinite in python)

• Fractions more complicated
  – number = (sign) * $1.(2^{-fraction}) \times 2^{(exponent - 127)}$
ASCII and Unicode

• On consumer laptops, memory is organized into bytes
  – A byte is eight bits
  – A bit is a single on/off (or 0/1, or true/false) value

• For many years, ASCII (American Standard Code for Information Exchange) has been the world standard
  – Uses seven bits to represent each character
  – Enough to represent all characters on a standard keyboard
  – Companies could use the eighth bit to add more characters

• Unicode is a newer standard
  – Can encode ASCII characters but also uses multiple bytes for additional characters
Casts

- To cast is to take a value of one type and return the corresponding value of some other type
- `int(x)` casts a string, float, or boolean `x` to an integer
- `float(x)` casts a string, integer, or boolean `x` to a float
- `str(x)` returns the value of `x` as a string
- `bin(int)` returns a string representing the binary value of the integer `int`
- `chr(int)` returns the character represented by the Unicode value `int`
- `ord(ch)` returns the integer value of the Unicode character `ch`
Equality

• Numerically not a good idea to compare floating numbers exactly
  – It might or might not work depending on the previous computations and what number is stored exactly

• Mostly safe to compare integers
  – Beware of loss of precision, however
  – IDLE example: precision. py

• If fractions are not required, better to use integers
  – Always keep numeric issues in mind
Approximate Equality

• What if you want to determine whether two floating point numbers are approximately equal?

• Test if difference is less than some small epsilon
  – \( \text{abs}(x - y) < \text{epsilon} \)
  – Different definitions of “small” depending on application

• Can also test if ratio is close to 1
  – \( \text{abs}(x/y - 1.0) < \text{epsilon} \)
  – \text{epsilon} doesn’t need to be adjusted for magnitude of x and y
  – Will fail if \( y = 0 \)
  – Be careful: \( (y == 0) \text{ or } (\text{abs}(x / y - 1.0)) < 0.00001 \text{ not a fix} \)
  – If \( y = 0 \), python won’t evaluate second part
Precedence

• 2 + 3 * 4 is 14, not 20
• This is because multiplication has higher precedence than addition
• Here’s what you should remember about precedence:
  – Exponentiation (**) has highest precedence
  – Unary operators have higher precedence than the related binary operators in the same family
  – By “family” I mean arithmetic, logic, or bitwise operators, so not x or y means the same as (not x) or y
  – Multiplication, division, and “and” operators have higher precedence than additions, subtraction, and “or” operators
  – x or y and z means the same as x or (y and z)
• For everything else, use parentheses to clarify your meaning, even if they aren’t needed
  – Make sure you use (parentheses) instead of [brackets] or {braces}
  – Good style to add spaces between operators, like in a natural language
Comparison chaining

- In Python, chain comparisons are allowed
  - \( x < y < z \) will work as expected
- In other languages, this will fail unless \( z \) is boolean because \( x < y \) would be boolean

- Feel free to use it but I’d stay away
  - You will have trouble when you switch to Java
Shorthand arithmetic

- $x += y$ is the same as $x = x + y$
- Same goes for $*=$, $-=$, etc.