SOME BUILT-IN TYPES, LITERALS AND EXPRESSIONS

Python Programming — Auburn University

BOOLEANS

- Literals: `True, False` (reserved words)
- Constructor: bool()
- Boolean operators: and, or, not. Examples:
  - `not True and True == False`
  - `not False and True == True`
  - `not (False or False) == True`

INTEGERS

- Signed numeric values without fractional part.
- Not fixed storage size
  - `30414093201713378043612608166064768844377641568960512000000000000` is a perfectly valid Python int.
- Examples of int literals: 5, -37
- Lots of literal syntax options: `0b1101` treats `1101` as a binary number and produces the integer `13`.
  - See https://www.python.org/dev/peps/pep-3127/
- Can also be created by the int() “constructor” — more later.

FLOATS

- Standard 64-bit floating point values (like Java double type). Most platforms use IEEE-754 standard representation.
- *Approximately* 15 (base-10) digits of precision.
- Range: $\pm 1.7 \times 10^{\pm 307}$.
  - For perspective: our universe is approximately 3.4×10^{17} seconds old.
  - Radius of Hydrogen nucleus: 10^{-15} meters
- Literals: 15.182736, -3.121, -8e+5, 7e-13
  - note 7e-13 = 7×10^{-13}
- float() constructor — later

COMPLEX

- Real and imaginary parts are floats.
    - note: Python uses “j” to represent the principle square root of -1 (common in engineering). You might be more used to “i”.
- We won’t have any use for complex numbers in this course ;)
- Literals: `3+2j`

NUMERIC OPERATIONS

- All numeric types support standard arithmetic, as in Java.
- Exponentiation is **: so 2**5 == 32.
- Notes on integer division:
  - / is floating-point division so 2/3 == 0.666…
  - // is “floored” division (like / in Java) so 2//3 == 0 and 9//2 == 4 etc.
  - % is modulus
  - Demo
- Integers also support bit-wise operations (|, &, <<, >>, ~). See reference.

NUMERIC OPERATORS CONTINUED

- Comparisons: <, >, <=, >=, ==, !=. Example:
  - `5 < 3 == False`
  - `5 > 3 == True`
  - `3 != 2 == True`
- Identity comparisons: `is`, `is not` — more later

NONETYPE

- NoneType is a type with a single instance: `None`
- Like `null` in Java
- Compare with `is` and `is not` operators:

```
if a is None:
    doSomething()
```

STRING

- Literals:
  ```
  “hello”
  ‘hello’
  “““hello”””
  ```
  all produce the same string.

- The triple quote syntax permits entering a multi-line string in a natural way:
  ```
  “““This
  is a valid
  string”””
  ```

- All syntaxes support “escaped” special characters such as `\n` for a newline, `\t` for tab etc. Note that REPL-printer prints these using the escape sequence so you’ll need to call the `print` function to see the actual string: Demo.

- Many other string-literal syntaxes available.
  - One of the most common is the formatted string literal that begins with `f”` which permits expression substitution. We’ll cover this once we’ve seen variables.

USE THE REFERENCE, LUKE!

https://docs.python.org/3/library/stdtypes.html