Primitive Data

Table of contents

Outline . . . 1
Learning objectives . . . 1
Primitive data . . . 2
Key results . . . 2
Converting between types — "albeit imperfectly" . . . 3
Exercise: string comparison . . . 3
Summary . . . 4
Next lecture . . . 4

See csse1001[1] for course logistics — this note covers Lecture 2B's technical content.

Outline

- Primitive data types built into Python
- Booleans, integers, strings, and tuples
- Converting between types

Learning objectives

1. There are more built-in types besides integers that can form expressions.
2. These types are immutable.
3. We can convert between types, albeit imperfectly.

[1] courses/csse1001/csse1001.pdf

Primitive data

The types "built-in" to Python by default are called **primitive data**. They are:

- booleans,
- integers,
- float (which we skip),
- strings, and
- tuples.

Booleans, integers and strings each come with a conversion function that shares the type's name: `bool`, `int` and `str`.

See python-primitive-data-types[2] for the full definitions, operators, and worked examples — comparison operators and truthy/falsy for booleans; unbounded precision for integers; concatenation, comparison, indexing/slicing and immutability for strings; and construction, indexing/slicing, immutability and packing/unpacking for tuples.

Key results

A handful of results carry everything else:

- **Booleans are numbers when you need them to be.** `True` has integer value `1`, so `3 + (2>1) + 3` is `7`. Comparison has *lower* precedence than arithmetic, so `3 + 2 > 1 + 3` is `(3+2) > (1+3)`.
- **Falsy is whatever acts as zero.** `bool(0)`, `bool("")` and `bool(())` are all `False`; everything else in these types is truthy. A space is not empty — `len(" ")` is `1` and `bool(" ")` is `True`.
- **Strings and tuples are the same shape.** Both are ordered, so both index from zero, index from the back with negatives, slice, concatenate with `+`, and repeat with `*`. Both are immutable, so both reject item assignment with the same `TypeError` — `'str' object does not support item assignment` and `'tuple' object does not support item assignment`. A tuple slices with the same `[start:stop:step]` syntax as a string, e.g. `xs[0:4]`.
- **The singleton tuple needs its comma.** `(1)` is the integer `1`; `(1,)` is the tuple. `(1) + (2,3)` is the common error.

[2] courses/csse1001/python-primitive-data-types.pdf

Converting between types — "albeit imperfectly"

Each of `bool`, `int` and `str` doubles as a conversion function, which gives six conversions among those three types. Only one of the six can fail — which is what the third learning objective is about.

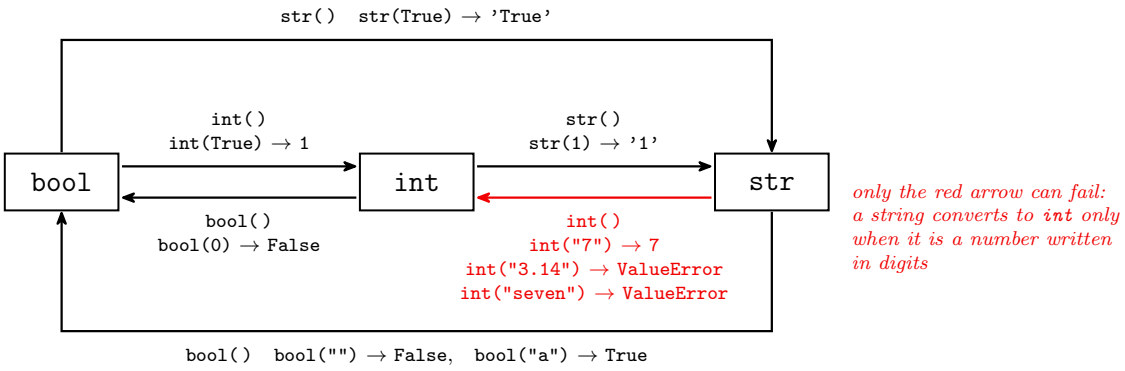

Figure 1: The six conversions among `bool`, `int` and `str`, each with an example call and its result. Only the red arrow, `str` to `int`, can fail. `float` is left out.

Python will not do the conversion for you: `3 + "7"` is a `TypeError`, and you have to say whether you meant `str(3) + "7"` (`'37'`) or `3 + int("7")` (10).

Exercise: string comparison

An exercise on string comparison:

```
def string_less_than(cs: str, ds: str) -> bool:
    """
    Restrict yourself to comparing integers only.
    Return true only when cs < ds.
    >>> string_less_than("aba", "ab")
    False
    >>> string_less_than("aZ", "aa")
    True
    """
```

"Restrict yourself to comparing integers only" rules out `<` on the strings themselves and points at `ord`. The two doctests are the two cases that catch a naive attempt — a prefix loses to its extension (`"aba" < "ab"` is `False`), and an uppercase letter beats a lowercase one, since `ord("Z")` is 90 and `ord("a")` is 97.

Summary

We have booleans, integers, strings, and tuples. Strings and tuples are **indexed** and can be **sliced**. We can convert between the types (imperfectly).

Next lecture

1. 2025-08-11-functions[3] — Functions (sequence).
2. 2025-08-11-sequence-selection-and-iteration[4] — If-statements (selection).

[3] courses/csse1001/2025-08-11-functions.pdf
[4] courses/csse1001/2025-08-11-sequence-selection-and-iteration.pdf