Types

Dr. Giuseppe Maggiore

Hogeschool Rotterdam
Rotterdam, Netherlands

Introduction

Lecture topics

- We introduce the Python type system
- Numbers
- Boolean values
- Arithmetic and boolean expressions

Python type system basics

Introduction

- Is everything an integer number?
- Yes and no

Python type system basics

Everything is an integer number

- For the CPU everything is a string of bits
- So yes, everything is (*almost*[a]) an integer number
- Complex data structures like a GUI, a 3D model, a picture, etc. are made up of collections of numbers

[a] also floats are recognized by the CPU

Python type system basics

Everything is an integer number

- Low-level languages expose this view
- Everything is encoded with numbers
- It can become quite messy

Python type system basics

Not everything is an integer number

- For the programmer, there exist different kinds of values
- So common and useful that Python offers them out of the box
- Even if the CPU does not manipulate them directly

Python type system basics

Kinds of values

- Python has a **type system**
- Variables have different **data types**, often shortened to **types**
  - `Integer` numbers
  - Rational (`floating point`) numbers
  - `Boolean` truth values
  - `Strings` of text

Python type system basics

Integers

- Numbers without dot/comma
- 0, 100, −500, ...

Python type system basics

Integers

- Typical arithmetic operations on numbers (**not in Python 3**)
- 3 + 5 = 8, 5 / 2 = 2, 40 * 5 = 200, ...

Python type system basics

Floating points

- Numbers with dot/comma
- `0.0`, `2.5`, `10.0e3`, `3.1e-5`, `-.1e-5`, ...

Python type system basics

Floating points

- Typical arithmetic operations on numbers
- `5.0 / 2.0 = ?`, `10.0e3 / 0.1 = ?`, `3.1e-5 + 1.0e5 = ?`, ...
- **Can you guess the results?**

Python type system basics

Floating points

- Typical arithmetic operations on numbers
- `5.0 / 2.0 = ?`, `10.0e3 / 0.1 = ?`, `3.1e-5 + 1.0e5 = ?`, ...
- **Can you guess the results?**
- `5.0 / 2.0 = 2.5`, `10.0e3 / 0.1 = 10.0e4`, `3.1e-5 + 1.0e5 = 100000.000031`, ...

Python type system basics

Conversion to and from floating point

- Integers can be converted to floating points with `float(n)`
- Floating points can be converted to integers with `int(n)`

Python type system basics

Conversion to and from floating point

- `int(2.5) = ?`, `float(3) = ?`
- **Can you guess the results?**

Python type system basics

Conversion to and from floating point

- `int(2.5) = ?`, `float(3) = ?`
- **Can you guess the results?**
- `int(2.5) = 2`, `float(3) = 3.0`

Python type system basics

Conversion to and from floating point

- Floating points can lose their decimal values
- They stay float's, but always end in `.0`
- `math.floor(n)` truncates the tail
- `math.ceil(n)` fills the tail and increases to the next unit

Python type system basics

Conversion to and from floating point

- `floor(2.5) = ?`, `ceil(2.5) = ?`
- **Can you guess the results?**

Python type system basics

Conversion to and from floating point

- `floor(2.5) = ?`, `ceil(2.5) = ?`
- **Can you guess the results?**
- `floor(2.5) = 2.0`, `ceil(2.5) = 3.0`

Python type system basics

Conversion to and from floating point

- Some conversions happen automatically
- Python operations try to preserve information
- `5 / 2.0 = 2.5`, and `5` is converted to `5.0` right before the division

Python type system basics

Python 3 integer division

- The new version of Python has a new integer division: it always converts to float
- It is **very different** from most other programming languages
- `5 / 2 = 2.5`

Python type system basics

Python 3 integer division

- Traditional integer division is now //
- `5 // 2 = 2`

Python type system basics

Boolean values

- Truth values
- `True`, `False`

Python type system basics

Boolean values

- Logical operators on truth values
  - `&` for `and`
  - `|` for `or`
  - `not`

Python type system basics

Boolean values

- Comparison operators on numeric values
  - >
  - <
  - ==
  - >=
  - <=

Python type system basics

Boolean values

- `5.0 > 2.0 = ?`, `(3 > 4) | (5 == (3 + 2)) = ?`, `True & False = ?`, ...
- **Can you guess the results?**

Python type system basics

Boolean values

- `5.0 > 2.0 = ?`, `(3 > 4) | (5 == (3 + 2)) = ?`, `True & False = ?`, …
- **Can you guess the results?**
- `5.0 > 2.0 = true`, `(3 > 4) | (5 == (3 + 2)) = False`, `True & False = False`, …

Python type system basics

String values

- Text
- `"Hello!"`, `"Hello world!"`, `""`, ...

Python type system basics

String values

- String literals are sequences of characters, on a single line, between double " or single ’ quotes
- Some characters do not fit this description
- We need special markings for such characters

Python type system basics

String values

- \' for single quote
- \" for double quote
- \a for ASCII Bell (BEL)
- \b for ASCII Backspace (BS)
- \f for ASCII Formfeed (FF)
- \n for ASCII Linefeed (LF)
- \r for ASCII Carriage Return (CR)
- \t for ASCII Horizontal Tab (TAB)
- \v for ASCII Vertical Tab (VT)
- \\ for newline

Python type system basics

String values

- The most common operator is string concatenation
- `"Hello" + "\n" + "world" + "\n" + "on" + "\n" + "different" + "\n" + "lines"`

Type restrictions

Operations, types, and restrictions

- Not all operations are allowed on all possible variable types
  - Some operations are allowed (integer addition)
  - Some operations are not allowed (string division)
  - Some operations change meaning (addition of integers versus concatenation of strings)

Type restrictions

Operations, types, and restrictions

- Examples of allowed operators
  - Addition, subtraction, division, multiplication, etc. between numbers
  - Concatenation between strings
  - Multiplication of strings and integers
  - Arithmetic comparison between numbers or strings
  - Conjunction, disjunction, negation[a] between booleans
  - Treating integers as booleans (`1=True`, `0=False`)
  - Treating strings as booleans (anything else=`True`, `""=False`)

[a] `and`, `or`, `not`

Type restrictions

Operations, types, and restrictions

- Examples of not-allowed operators
  - Most arithmetic operations on strings and non-strings (`"Hello" + True`)
  - Most boolean operations on strings and non-strings (`"Hello" & True`)

Type errors

Not-allowed operators generate *type errors*

```
Traceback (most recent call last):
  File "C:\Users\Giuseppe\Desktop\DEV␣I␣samples\
      DEV_I_samples.py", line 8, in <module>
    print("Oh␣noes,␣a␣bug!" + 4)
TypeError: cannot concatenate ’str’ and ’int’
    objects
```

Type restrictions

Operations, types, and restrictions

- Variables may change type in Python
- An integer variable becomes later on a string variable
- This is allowed, but dangerous
- A variable should never lose reasonable meaning
- Many type errors stem from *changes in meaning*, connected with *changes in type* of a variable

This is it!

The best of luck, and thanks for the attention!