CSC110 Lecture 2: Representing Data

David Liu, Department of Computer Science, University of Toronto

Navigation tip for web slides: press ***?*** *to see keyboard navigation controls.*

Welcome back!
Take a few minutes to go around your table and introduce yourself 👋

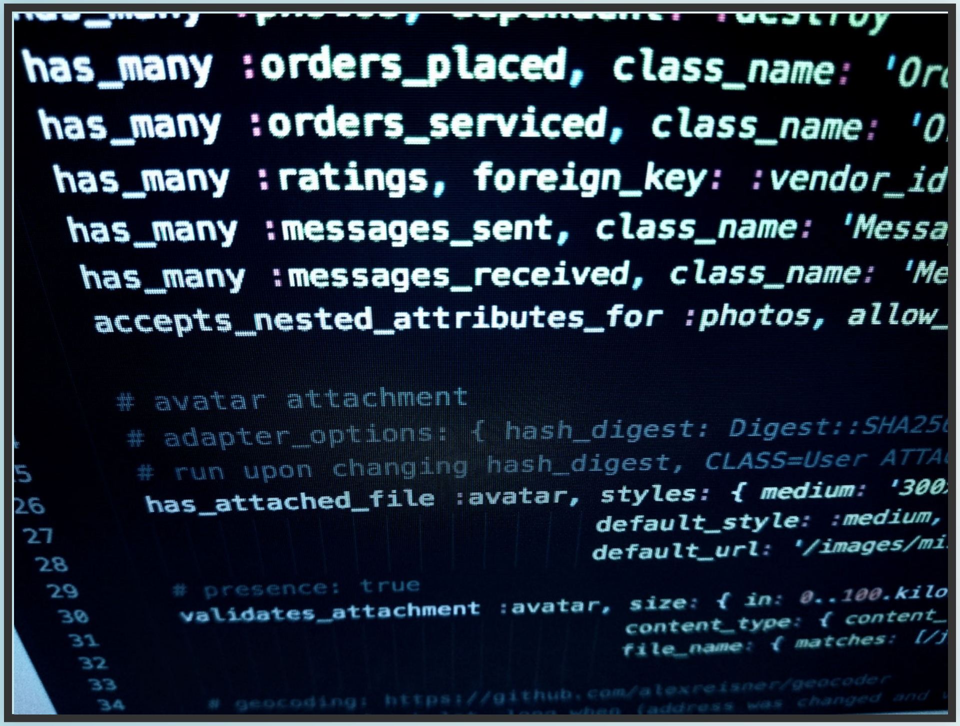
Some introductions (Course Team)

Also see our new Course Team page!

Quick announcements

- Recruiting a volunteer note-taker! (Campuswire post #5)
- U of T AI club: LearnAI course sign-up! (Campuswire post #6)
- (Hart House Play) Truth Values: Exploring Gender, Diversity and Unconscious Bias in STEM (Campuswire post #7)
- Additional Resources module posted on Quercus
    - Student Success Strategies: Note-taking

Recap



```
has_many :photos, dependent: :destroy
has_many :orders_placed, class_name: 'Or
has_many :orders_serviced, class_name: 'O
has_many :ratings, foreign_key: :vendor_id
has_many :messages_sent, class_name: 'Messa
has_many :messages_received, class_name: 'Me
accepts_nested_attributes_for :photos, allow_

# avatar attachment
# adapter_options: { hash_digest: Digest::SHA256
# run upon changing hash_digest, CLASS=User ATTA
has_attached_file :avatar, styles: { medium: '300
                                     default_style: :medium,
                                     default_url: '/images/mi

# presence: true
validates_attachment :avatar, size: { in: 0..100.kilo
                                      content_type: { content_
                                      file_name: { matches: [/j

# geocoding: https://github.com/alexreisner/geocoder
```

Terminology review (1.1, 1.2)

programming language: a language designed to allow humans to communicate instructions to a computer

program source code (or just **code**): text written in a programming language

Python: the programming language we'll be using in CSC110 and CSC111

Python interpreter: a program that takes code written in Python and executes the instructions the code contains

Python console: the interactive mode of the Python interpreter

Terminology review (1.2)

```
>>> 4 + 5
9
```

expression: a piece of Python code that produces a value when evaluated

- example: `4 + 5` produces `9`

literal: the simplest kind of expression, representing a value as written

- example: `9`; the `4` and `5` in `4 + 5`

operator: a symbol in code that represents a specific computation to perform

- example: the `+` in `4 + 5`

Basic Python data types (1.3, 1.4)

Data type	Description	Example Literals
`int`	integer data ($\mathbb{N}$, $\mathbb{Z}$)	`3`, `-999`
`float`	general numeric data ($\mathbb{Q}$, $\mathbb{R}$)	`3.5`, `-99.4`
`bool`	Boolean (True/False) data	`True`, `False`
`str`	Text data	`'CSC110'`, `'David is cool'`

Why are data types important to programmers?

1. Data types help us categorize real-world data and use them in our programs
2. Each data type determines what operations we can perform on a piece of data

Basic Python operations (1.3, 1.4)

Type	Operations
`int`, `float`	Arithmetic (e.g. `+`, `*`), comparisons (e.g. `==`, `<`)
`bool`	`and`, `or`, `not`
`str`	`==`, `+`, `in`, indexing (`s[...]`)

Quick check

What do each of the following Python expressions evaluate to?

```
>>> 2 ** 3  # reminder: ** means "to the power of"
```

```
>>> 2 + 3.0
```

```
>>> not (5 * 2 > 3)
```

```
>>> 'hello' + 'CSC110'
```

```
>>> 'hello'[1]
```

Today’s learning goals

In this lecture, you will learn to:

1. Define and identify different types of collection data.
2. Represent these collection data types in the Python programming language (using the Python console).
3. Perform simple operations on collection data in Python.
4. Use variables to refer to values in a Python program.
5. Use a memory model diagram to keep track of variables in a Python program.

Three collection data types

Set data

A **set** is a collection of zero or more distinct values, where order does not matter.

- Example: $\{1, 2, 3\}$

In Python, we represent sets using the `set` data type.

`set` literals are written with curly braces, e.g. `{1, 2, 3}`.

Set operations

`set1 == set2` : check whether two sets are equal (order doesn’t matter!)

`x in my_set` : check whether a value is an element of a set (like ∈)

List data

A **list** is a sequence of zero or more values that may contain duplicates.

- Example: $[1, 2, 3]$
- sequence means that the order of values matters

In Python, we represent lists using the `list` data type.

`list` literals are written with square brackets, e.g. `[1, 2, 3]`.

List operations

Like sets, lists are collections of elements.

- `==` and `in` work with lists!

Like strings, lists are sequences.

- `+` (concatenation) and `[_]` (indexing) work with lists!

Sets vs. lists

Use a **set** when:

- your data cannot contain duplicates, and
- order does not matter in your data

Use a **list** when:

- your data may contain duplicates, or
- order matters in your data

Mapping data

A **mapping** is a collection of **association pairs**. Each pair consists of a **key** and **associated value**.

- Example: $\{\text{‘Toronto’} : 2.8, \text{‘Ottawa’} : 1.0\}$

In Python, we represent mappings using the `dict` data type.

`dict` literals are written with curly braces and colons separating keys and associated values, e.g. `{'Toronto': 2.8, 'Ottawa': 1.0}`.

Mapping operations

`dict1 == dict2`: check whether two mappings are equal

`key in my_dict`: check whether a given key is in the mapping

`my_dict[key]`: produce the associated value in the mapping for the given key

Exercise 1: Data Types in Python

Our first exercise!

Open up today’s exercise webpage (on Quercus) and complete **Exercise 1: Data Types in Python**.

Tip: you can print the webpage to PDF if you want to write on it directly.

Homogeneous collections

In Python, collections can contain values of different types.

Examples: `{1, 'hi', True}` or `{'david': 3, 4: True}`

- A `set`/`list` is **homogeneous** when its elements all have the same type
- A `dict` is **homogeneous** when its keys all have the same type, and associated values all have the same type

A collection that is not homogeneous is called **heterogeneous**.

In CSC110/111, almost all collections we'll work with will be **homogeneous**.

Empty collections

It is sometimes useful to represent collections that have **no** elements. In Python:

Collection type	**Empty collection**
`set`	`set()`
`list`	`[]`
`dict`	`{}`

Variables

Storing values

A variable is a piece of code that refers to a value.

Created using **assignment statements**:

```
<variable> = <expression>
```

Executing an assignment statement

```
<variable> = <expression>
```

1. Python interpreter evaluates `<expression>`.
2. Python interpreter assigns the resulting value to `<variable>`.

```
>>> x = 10 + 30
```

```
>>> x
40
```

Statements vs. expressions

A **statement** is a piece of code representing an instruction to the computer.

Statement type	Instruction
expression	evaluate this
assignment	evaluate the right-hand side and assign the value to the left-hand side

Every expression is a statement, but not every statement is an expression.

Keeping track of variables

```
a = 3
b = 7 * a
c = [1, a, b]
```

```
d = c[0] + a
```

Variable	Value
a	3
b	21
c	[1, 3, 21]
d	4

Value-based memory model

A **memory model** is a structured way of representing variables and data in a program.

Our previous example of a table of values is a value-based memory model.

Demo: PyCharm’s “Special Variables” view

Exercise 2: Variables

Summary

The seven main Python data types

Data type	Description	Operations
`int`, `float`	Numeric data	Arithmetic (e.g. +), comparisons (e.g. ==, <)
`bool`	Boolean (True/False) data	`and`, `or`, `not`
`str`	Text data	==, +, `in`, indexing (`s[...]`)
`set`	Collection, no duplicates, no order	==, `in`
`list`	Collection, duplicates allowed, order matters	==, +, `in`, indexing (`s[...]`)
`dict`	Collection of association pairs	==, `in`, key lookup (`d[...]`)

Today, you learned to...

1. Define and identify different types of collection data.
2. Represent these collection data types in the Python programming language (using the Python console).
3. Perform simple operations on collection data in Python.
4. Use variables to refer to values in a Python program.
5. Use a memory model diagram to keep track of variables in a Python program.

Homework

- Review today's lecture
  - Course Notes: 1.3, 1.4, 1.5, 1.6
  - Additional exercises (bottom of today's handout)
  - Office hours today: 2-4pm (BA 4290), 6-7pm (online)
- Reading ahead:
  - Tuesday's class: 1.7, 2.1, 2.2
  - Thursday's class: 2.4, 2.7
  - Tutorial: 1.8

Course Syllabus, Software Installation Guide, Welcome Survey

Tip of the day

As CS students, you have access to the department’s **computer labs and printers** on the 2nd/3rd floor of Bahen!

See https://www.teach.cs.toronto.edu/ for details!

DETERMINATION.
ICANHASCHEEZBURGER.COM