Turtle

Nicolò Pratelli

Istituto di Scienza e Tecnologie dell’Informazione
Consiglio Nazionale delle Ricerche – Pisa

A.A. 2024-2025

Practical information for seminars

- **Tutor**: Nicolò Pratelli, Graduate Fellow at the ISTI-CNR
  nicolo.pratelli@isti.cnr.it
- **Seminar**: a brief **summary** of a topic followed by individual exercises
- **Exercises**: will be published on the course Moodle page during the lesson
  https://elearning.di.unipi.it/course/view.php?id=105
- Completing exercises is **strongly suggested**, but not mandatory.
- Once you have completed the exercises, please reach out to us for feedback to better understand any potential mistakes.
- Solutions will be published on Moodle after the lesson.

RDF

- The Resource Description Framework (RDF) is a family of World Wide Web Consortium (W3C) specifications designed as a semantic data model

http://www.w3.org/TR/rdf11-primer

- The RDF data model is based upon the idea of making **statements** about **resources** in form of *subject–predicate–object*, known as **triples**.
- The subject an the object denotes the resource.
- The predicate denotes traits or aspects of the resource, and expresses a **relationship** between the subject and the object.

Turtle

- **Terse RDF Triple Language** (**Turtle**) is a **concrete syntax** and file format for expressing data in the Resource Description Framework (RDF) data model.

https://www.w3.org/TR/turtle/

- Turtle syntax is similar to that of SPARQL, an RDF query language.
- In this course we will use Turtle for storing RDF data, but have been developed other syntaxes:

N-Triples, JSON-LD and RDF/XML.

Editors for Turtle

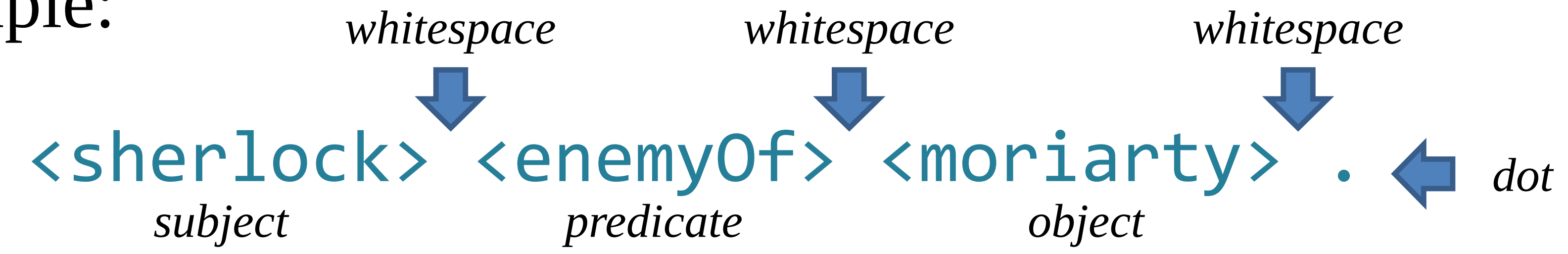
You can use any text editor to write Turtle but it is a good idea to use an editor that supports syntax highlighting, such as:

- Notepad++ (https://notepad-plus-plus.org) using the Turtle syntax
- Visual Studio Code (https://code.visualstudio.com) using the Turtle extension

Triples in Turtle

In Turtle, a **triple** is a sequence *subject–predicate–object* separated by **whitespace** and terminated by a dot (.)

Example of triple:



```
<sherlock> <enemyOf> <moriarty> .
```

This triple states that Sherlock Holmes is enemy of professor Moriarty (which is true in the world of Sherlock Holmes novels)

The terms `<sherlock>`, `<enemyOf>`, and `<moriarty>` are **IRIs**

```
<http://ex.org/sherlock> <http://ex.org/enemyOf> <http://ex.org/moriarty> .
```

subject *predicate* *object*

IRIs

IRIs (Internationalized Resource Identifiers) are strings of characters used to identify resources in RDF.

Example of IRI:

```
<http://example.org/sherlock>
```

IRIs are a superset of **URIs (Uniform Resource Identifiers)**, supporting Unicode characters instead of just ASCII

URIs are a superset of **URLs (Uniform Resource Locators)**, used to identify web resources such as a webpage, and also of **URNs (Uniform Resource Names)**, another type of persistent identifier that is not necessarily resolvable via HTTP

IRIs in Turtle

In Turtle, IRIs can be written in three ways:

1) **Absolute IRI**

```
<http://example.org/sherlock>
```

2) **Relative IRI**

```
<sherlock>
```

3) **Prefixed name**

```
ex:sherlock
```

Absolute and relative IRIs use angle brackets. Prefixed names don't use brackets and always contain a colon

Absolute IRIs

In this example shows 2 triples define Sherlok Holmes and Dr. Moriarty as Person and 1 triple states Sherlok Holmes is enemy of Dr. Moriarty.

```
<http://example.org/sherlock> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.org/Person> .
<http://example.org/moriarty> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.org/Person> .
<http://example.org/sherlock> <http://another.namespace/enemyOf> <http://example.org/moriarty> .
```

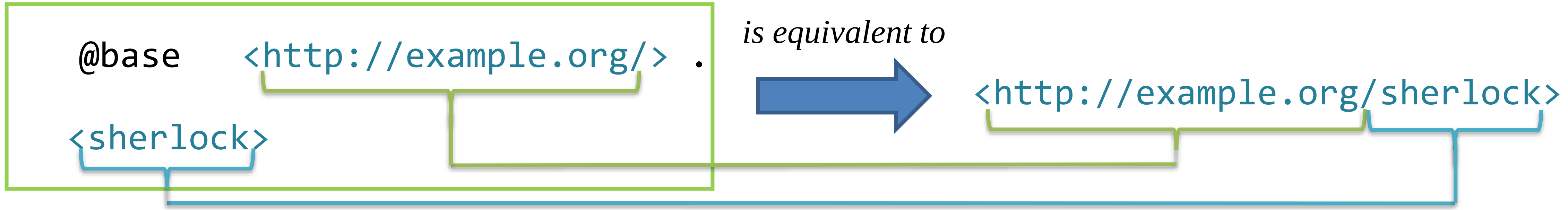
Absolute IRIs

In this example shows 2 triples define Sherlok Holmes and Dr. Moriarty as Person and 1 triple states Sherlok Holmes is enemy of Dr. Moriarty.

```
<http://example.org/sherlock> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.org/Person> .
<http://example.org/moriarty> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.org/Person> .
<http://example.org/sherlock> <http://another.namespace/enemyOf> <http://example.org/moriarty> .
```

Relative IRIs

A relative IRI makes reference to a base IRI, and they can be combined to obtain the absolute IRI



```
@base   <http://example.org/> .
<sherlock>
```

is equivalent to

```
<http://example.org/sherlock>
```

Most Turtle documents contain only one @base directive, generally at the top of the document

The base IRI usually ends with a slash “/” or a hash “#” character. If you forget this character, the IRI will not work correctly

Prefixed Names

A prefixed name is composed of a prefix label and a local part, separated by a colon, without angle brackets

```
@prefix ex: <http://example.org/> .

 ex:sherlock
```

is equivalent to → `<http://example.org/sherlock>`

Each Turtle document may contain multiple @prefix directives, generally at the top of the document.

The prefix label may also be empty. This empty prefix is called default prefix

```
@prefix : <http://example.org/> .

 :sherlock
```

is equivalent to → `<http://example.org/sherlock>`

Prefixed Names Only

```
@prefix    : <http://example.org/> .
@prefix  an: <http://another.namespace/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

:sherlock  rdf:type  :Person .
:moriarty  rdf:type  :Person .
:sherlock  an:enemyOf  :moriarty .
```

Relative IRIs & Prefixed Names

You can use both relative IRIs and prefixed names at the same time. In the following example we are importing rdf:type from the RDF vocabulary, and we are also using a default prefix

```
@base <http://example.org/> .
@prefix : <http://another.namespace/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

<sherlock>  rdf:type <Person> .
<moriarty>  rdf:type <Person> .
<sherlock> :enemyOf <moriarty> .
```

In general, it is your choice which notation to use

Literals in Turtle

Literals are used to identify simple values such as strings, numbers, dates, etc.

In Turtle, literals are always enclosed in quotation marks (single, double or a combination of them)

```
@prefix : <http://example.org/> .

:sherlock :name "Sherlock" .
```

In this example, we are stating that the name of the resource `:sherlock` is the string of characters `"Sherlock"`.

Datatyped Literals

Literals are composed of a lexical form and a datatype, separated by two carets ^^

```
@prefix : <http://example.org/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
:sherlock :name "Sherlock"^^xsd:string .
:sherlock :age  "60"^^xsd:integer .
```

If the datatype is omitted, it is automatically interpreted as `xsd:string` (a simple string)

is equivalent to

```
:sherlock :age "60"
```

$\Rightarrow$

```
:sherlock :age "60"^^xsd:string .
```

Language Tags

String literals may also have a language tag, expressing the language of the string

The language tag is preceded by an @ symbol

```
@prefix : <http://example.org/> .

:sherlock :name "Sherlock"@en .
```

The language tag and the datatype **can’t** be expressed at the same time. When there is a language tag, the datatype is interpreted as `rdf:langString` (a language-tagged string)

Boolean Literals

Boolean literals are expressed using true or false (all lowercase) and have datatype `xsd:boolean`. You don't need to specify the datatype:

```
@prefix : <http://example.org/> .
:sherlock :hasAEnemy true .
```

Note: In this case you don't need to specify the `@prefix xsd:`

This is equivalent to:

```
@prefix : <http://example.org/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
:sherlock :hasAEnemy "true"^^xsd:boolean .
```

is equivalent to

`:sherlock :hasAEnemy "true"` ➡ `:sherlock :hasAEnemy "true"^^xsd:string .`

Time Literals

Time literals can be expressed using several XSD datatypes. We will use two of these:

- the first is `xsd:date` (expresses a date):

```
:seminar1 :hasDate "2024-09-30"^^xsd:date .
```

- the second is `xsd:dateTime` (expresses a date and a time):

```
:seminar1 :startsAt "2024-09-30 T14:15:00"^^xsd:dateTime .
```

Blank Nodes

Blank nodes in Turtle are expressed as `_:` followed by a blank node label, which is a sequence of characters

```
        _:somePerson :knows _:anotherPerson .
```

The label may contain letters, numbers, an underscore, and also a dot (except as the first or last character)

Blank nodes can also be denoted through nested elements

```
:semanticWebCourse :hasTeacher  [ :name "Valentina Bartalesi" ] .
```

is equivalent to

```
:semanticWebCourse :hasTeacher _:a

_:a :name "Valentina Bartalesi" .
```

Predicate Lists

Often, the same subject will be used in several triples. Turtle offers an abbreviated syntax for writing these triples, using a semicolon “;”

```
:sherlock :name "Sherlock" ;
          :knows :watson ;
          :enemyOf :moriarty .
```

This is equivalent to writing three full triples with subject `:sherlock`

```
:sherlock :name "Sherlock" .
:sherlock :knows :watson .
:sherlock :enemyOf :moriarty .
```

Note: writing three full triples you need to use a dot at the end of each line.

Object Lists

Often, the same subject and predicate is used in several triples. Turtle offers an abbreviated syntax for writing these triples, using a comma (,)

```
:sherlock :friendOf :lestrade ,
                    :watson .
```

This is equivalent to writing a predicate list with predicate `:friendOf` or two full triples with subject `:sherlock` and predicate `:friendOf`

```
:sherlock :friendOf :lestrade ;
          :friendOf :watson .
```

```
:sherlock :friendOf :lestrade .
:sherlock :friendOf :watson .
```

Instructions for Exercises

- For IRIs, use the fake namespace `<http://example.org/>` and invent your own names for the resources
- Remember that the **subject and predicate of a triple are always IRIs**, while the object can be an IRI or a literal
- Use `rdf:type` to define the type of the resource
- For representing textual content (such as names), use string literals. For dates, use the `xsd:date` datatype. For numbers, use the `xsd:integer` datatype
- Validate your code (http://ttl.summerofcode.be)
- The validation is highly recommended. If you have any issue, please write an e-mail.

Example Exercise

- Use the Turtle syntax to represent a set of facts about Italian poet **Dante Alighieri**: *type* of resource (Person), *name* (string), *date of death* (date), a *comment* (string), and state that he *is author of* the Divine Comedy (IRI)

Example Exercise: solution

```
@base <http://example.org/>    .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>   .


<dante> rdf:type <Person> ;
        <name> "Dante Alighieri" ;
        <comment> "Dante was a major Italian poet" ;
        <dateOfDeath> "1321-09-14"^^xsd:date  ;
        <isAuthorOf> <divineComedy>  .
```

Use the Turtle syntax to represent a set of facts about Italian poet **Dante Alighieri**:

- *type* of resource (Person),
- *name* (string),
- *date of death* (date),
- a *comment* (string),
- and state that he *is author of* the Divine Comedy (IRI)

Exercises

- Use the Turtle syntax to represent 2 separate sets of facts about each of the following resources:
- English computer scientist **Alan Mathison Turing**: type of resource, name, date of birth, date of death, the fact that he formalized the Turing Machine, using only relative IRIs.
- The **Apple** foundation of *April 1, 1976*: type of resource (event), date, and each of the three participants, using an object list for the participants