One language, more languages

Stefano Penge, may 2022

What is this lesson about

- When teaching CS and programming the teacher should ask him/herself: **which language** fits the best to my students competences and to their learning objectives?
- As we saw, there are other, less cognitive aspects, that could play an important role
  - personal preferences
  - previous experiences,
- There could also be practical aspects:
  - a language suggested by the text book already adopted by previous teacher
  - a default language that is supposed to be the best choice for that context
  - existing interpreters and compilers installed
  - availability and quality of network
- But it is always worth the effort to try to imagine if there are alternatives to the default options:
  - a specialized language that fits best for certain problem?
  - a simpler environment, just to sketch a program before writing a complete version?
  - a language the allow for compilation on a mobile device to test it more extensively?

Part I

More languages/1

- In some cases, there could be a good idea not to choose a single language, but to present multiple options
- When teaching natural languages, sometimes the teachers use different languages to show the relationship among languages, or else to make students aware of false friends
- This is based on the fact the everyone knows at least one natural language (mother tongue)

vetro	glass	verre
bicchiere	glass	verre
specchio	glass	miroir
occhiali	glasses	lunettes

Could you do the same for programming languages?

switch	select	case
case	when	when
def	function	defn
repeat	do	begin
until	while	end while

More languages/2

- There are a lot of resources on the web where one can compare implementation of the same algorithm in different languages
  - Rosetta Code

    http://rosettacode.org/wiki/Rosetta_Code
  - 99 bottles of beer

    http://99-bottles-of-beer.net/
  - Hello, World!
    http://helloworldcollection.de/

Ingredients.
72 g haricot beans
101 eggs
108 g lard
111 cups oil
32 zucchinis
119 ml water
114 g red salmon
100 g dijon mustard
33 potatoes

Method.
Put potatoes into the mixing bowl. Put dijon mustard into the mixing bowl. Put lard into the mixing bowl. Put red salmon into the mixing bowl. Put oil into the mixing bowl. Put water into the mixing bowl. Put zucchinis into the mixing bowl. Put oil into the mixing bowl. Put lard into the mixing bowl. Put lard into the mixing bowl. Put eggs into the mixing bowl. Put haricot beans into the mixing bowl. Liquefy contents of the mixing bowl. Pour contents of the mixing bowl into the baking dish.

Serves 1.

Ingredients.
72 g haricot beans
101 eggs
108 g lard
111 cups oil
32 zucchinis
119 ml water
114 g red salmon
100 g dijon mustard
33 potatoes

https://www.dangermouse.net/esoteric/chef.html

https://metacpan.org/release/SMUELLER/Acme-Chef-1.01

http://bogost.com/teaching/introduction_to_computational/

Method.
Put potatoes into the mixing bowl. Put **d**ijon mustard into the mixing bowl. Put **l**ard into the mixing bowl. Put **r**ed salmon into the mixing bowl. Put **o**il into the mixing bowl. Put **w**ater into the mixing bowl. Put zucchinis into the mixing bowl. Put **o**il into the mixing bowl. Put **l**ard into the mixing bowl. Put **l**ard into the mixing bowl. Put **e**ggs into the mixing bowl. Put **h**aricot beans into the mixing bowl. Liquefy contents of the mixing bowl. Pour contents of the mixing bowl into the baking dish.

Serves 1.

The project

- We want to write a simple program capable of creating little stories.
- There are several ways to design such a program (rewriting grammar, L-system)
- The simplest one is probably to use a template, like in the "cloze" quizzes.
- We will build a template, we will create different list of words and we will ask the computer to fill the template choosing randomly words from the lists.

Didactic method

- Before starting to write code, or even to design the program, there is a need to explore the domain through a series of questions and research activities:
  - Which is a structured text? How many types do you know? Give an example
  - How are structured texts produced by human writers?
  - Is it possible to write them automatically?
  - Look at this: https://www.plot-generator.org.uk/story/
  - What a program should know to write a text?
  - How can you tell if a text is produced by a human or by a software?
  - ...

Learning goals/1

- From the CS point of view:
  - different types of data structures
  - selecting data
  - string interpolation
  - function, recursion, …
  - categories of languages

Learning goals/2

- From the disciplinary point of view:
  - structured text (fairy tales, jokes, poems)
  - grammar categories
  - lexicon
  - generative rules

Difficulties

- Probably the only true difficulty is the one related to the use of the words:
  - how to keep some coherence among words (eg. knights and dragons should go together)?
  - how to keep the binding between a role and word through the story?
- The data structure can be dispersed or unique
- The main program could be a simple iteration on the template's lines; or could be a recursive function that "consumes" the template polling from the lists until the end

The template

1) There was once upon a time a <hero> striving for <kingdom>
2) Our <hero> was in quest for a <treasure>
3) After a long travel through <lands> the <hero> suddenly faced an enormous <opponent>
4) The <opponent> tried to <attack> the <hero>, but without success
5) Then it was the <hero> turn to <attack> the <opponent>; and so they went on and on for all the <duration>
6) Then, oh wonder!, the <opponent> was won and <subdued>
7) At the end of the adventure, the <hero> couldn't get the kingdom, because of <surprise>.

Word lists

1) hero (noun): knight, dwarf, singer, youtuber, ...
2) treasure (noun): crown, unique ring, cup of icecream, ...
3) kingdom (noun): Xanadu, Eurovision, Atlantide
4) lands (noun): desert, forest, supermarket, downtown, ...
5) opponent (noun): dragon, giant, Godzilla,
6) attack (verb infinitive): fire, disintegrate, hypnotize, ...
7) duration (noun): day, week, season
8) subdued (verb participle): tied to a tree, chained to a column, buried under a pyramid
9) surprise (noun): ...

Coherence

- All the magics stays in a clever choice of words
- One should define categories inside role lists to try to keep *vertical* coherence
- Each categories should contains standard substitutions for the blanks, but also *stranger* candidates that could mess up the story
- Depending on the context (age of the students, type of the text to be built) the more the story is surprising, the more the students will be intrigued

Judging

- The class could be divided in three teams, each of which using a different language
- Someone (other teachers? a group of students from another class? parents?) should be appointed as external judges to vote the best implementation (= the best story)
- Every teamwork could be voted on different parameters:
  - the most creative
  - the funniest
  - the scariest
  - …
- These are not judgment on the *code*, but on the *final result.*

Part II

Which languages

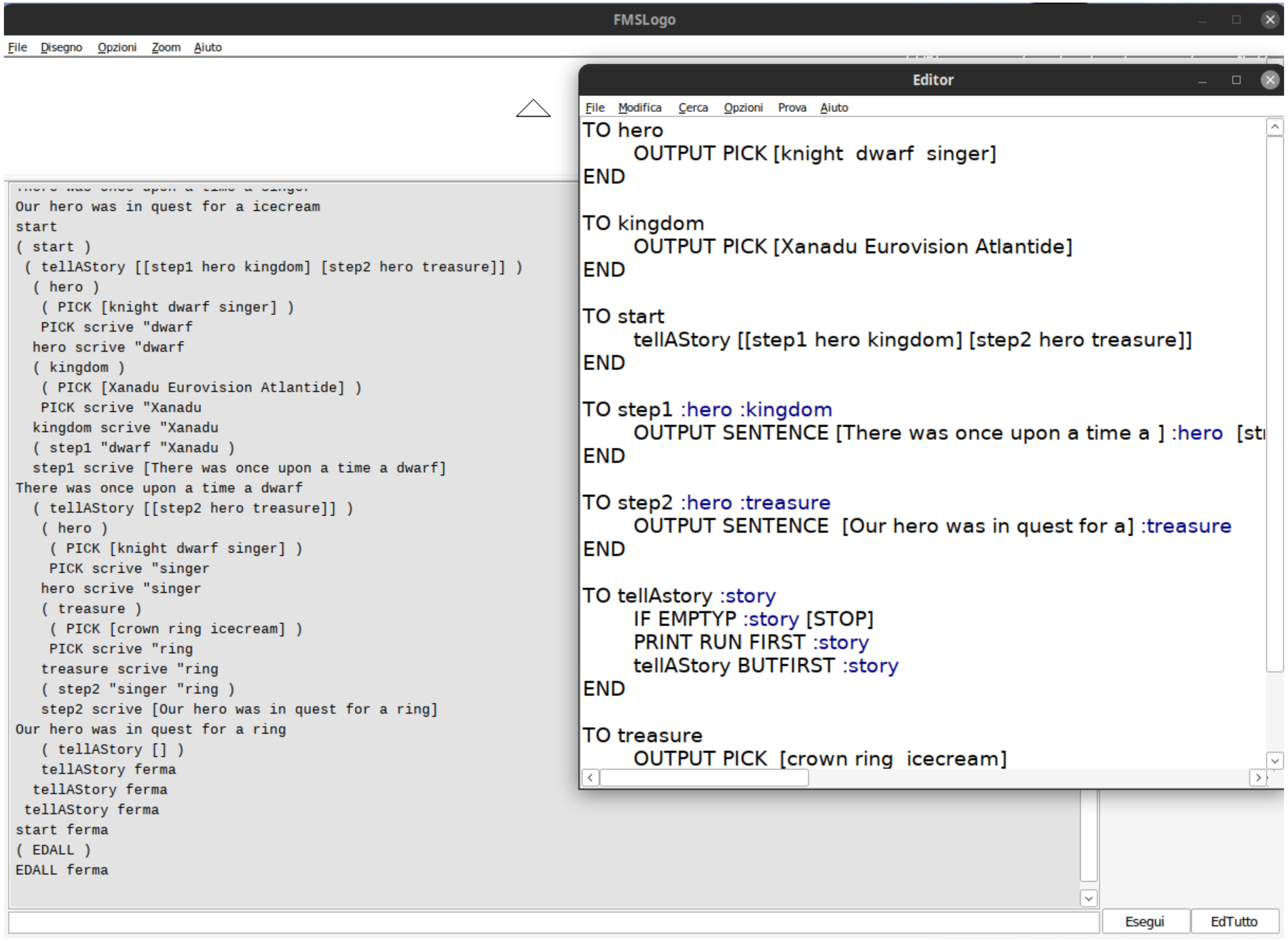
- Here follows three examples of implementation in different languages
- The idea is to use the main characteristics of each language and use the example to underline them
- The examples are inspired by those published in chapter 2 of Lingua, Coding e creatività, Anicia, 2017

```
TO hero
      OUTPUT PICK [knight  dwarf  singer]
END

TO kingdom
      OUTPUT PICK [Xanadu Eurovision Atlantis]
END

TO treasure
      OUTPUT PICK  [crown ring  icecream]
END

TO step1 :hero :kingdom
      OUTPUT SENTENCE [There was once upon a time a ] :hero  [striving for
a ] :kingdom
END

TO step2 :hero :treasure
      OUTPUT SENTENCE  [Our hero was in quest for a] :treasure
END
```

```
TO tellAstory :story
    IF EMPTYP :story [STOP]
    PRINT RUN FIRST :story
    tellAStory BUTFIRST :story
END

TO start
    tellAStory [[step1 hero kingdom] [step2 hero treasure]]
END
```

Logo

- The first one (Logo) is really basic: there are a series of identical functions that output a randomly chosen element and one function per story-step to bind the character to his role in the sentence
- A recursive main function is used to build the story
- The teacher could underline that there is no global variable
- Upgrade: could we use the MAP construct to apply the templates to the characters list?
- The Logo dialect used here is UCB; here is shown the Trace feature in FMSLogo for Windows


FMSLogo
File Disegno Opzioni Zoom Aiuto
Editor
File Modifica Cerca Opzioni Prova Aiuto
Esegui
EdTutto

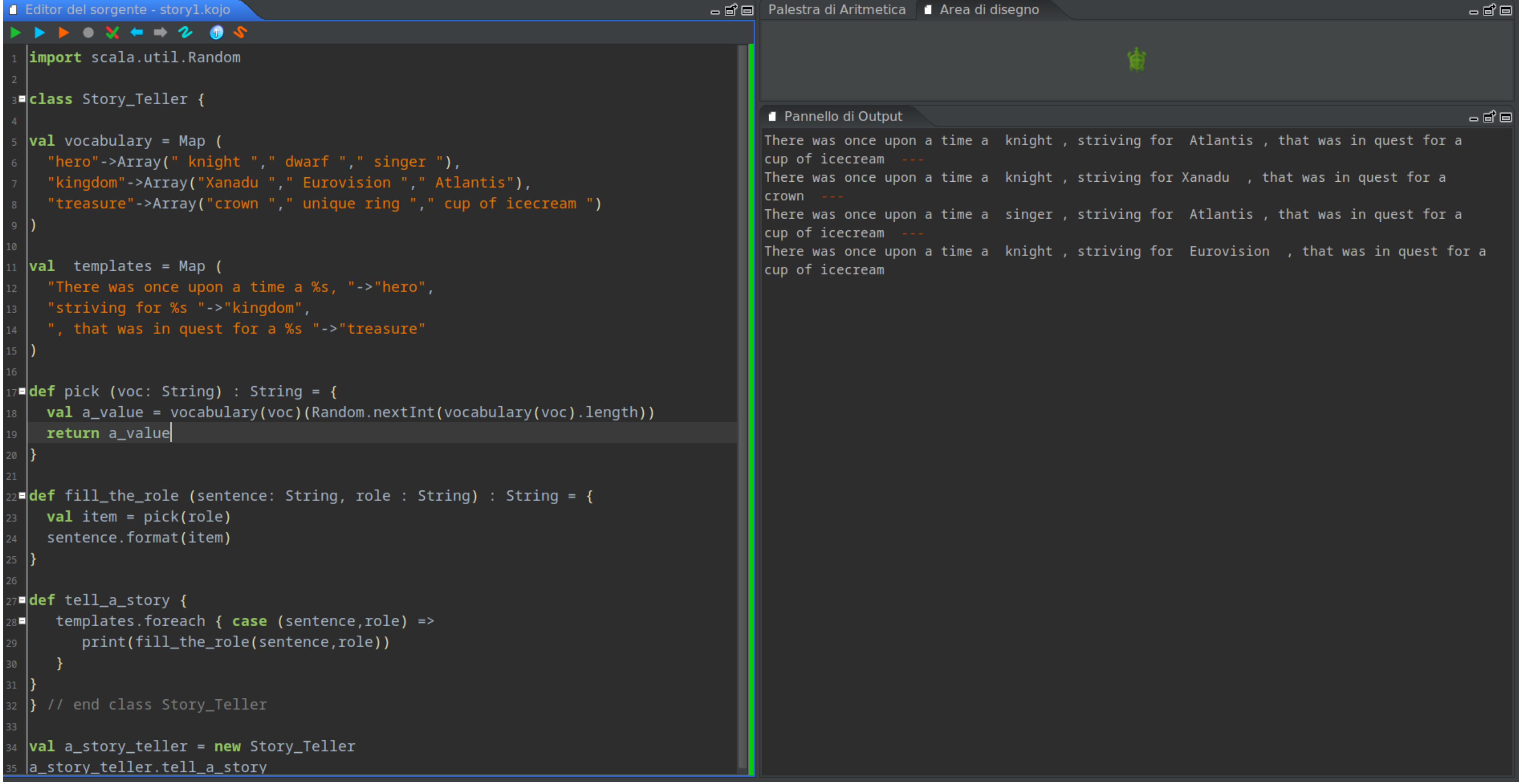
```
There was once upon a time a singer
Our hero was in quest for a icecream
start
( start )
 ( tellAStory [[step1 hero kingdom] [step2 hero treasure]] )
  ( hero )
   ( PICK [knight dwarf singer] )
   PICK scrive "dwarf
  hero scrive "dwarf
  ( kingdom )
   ( PICK [Xanadu Eurovision Atlantide] )
   PICK scrive "Xanadu
  kingdom scrive "Xanadu
  ( step1 "dwarf "Xanadu )
  step1 scrive [There was once upon a time a dwarf]
There was once upon a time a dwarf
  ( tellAStory [[step2 hero treasure]] )
   ( hero )
    ( PICK [knight dwarf singer] )
    PICK scrive "singer
   hero scrive "singer
   ( treasure )
    ( PICK [crown ring icecream] )
    PICK scrive "ring
   treasure scrive "ring
   ( step2 "singer "ring )
   step2 scrive [Our hero was in quest for a ring]
Our hero was in quest for a ring
   ( tellAStory [] )
   tellAStory ferma
  tellAStory ferma
 tellAStory ferma
start ferma
( EDALL )
EDALL ferma
```

```
TO hero
      OUTPUT PICK [knight  dwarf  singer]
END

TO kingdom
      OUTPUT PICK [Xanadu Eurovision Atlantide]
END

TO start
      tellAStory [[step1 hero kingdom] [step2 hero treasure]]
END

TO step1 :hero :kingdom
      OUTPUT SENTENCE [There was once upon a time a ] :hero  [st
END

TO step2 :hero :treasure
      OUTPUT SENTENCE  [Our hero was in quest for a] :treasure
END

TO tellAstory :story
      IF EMPTYP :story [STOP]
      PRINT RUN FIRST :story
      tellAStory BUTFIRST :story
END

TO treasure
      OUTPUT PICK  [crown ring  icecream]
```

```
import scala.util.Random

class Story_Teller {

val vocabulary = Map (
  "hero"->Array(" knight "," dwarf "," singer "),
  "kingdom"->Array("Xanadu "," Eurovision "," Atlantis"),
  "treasure"->Array("crown "," unique ring "," cup of icecream ")
)

val  templates = Map (
  "There was once upon a time a %s, "->"hero",
  "striving for %s "->"kingdom",
  ", that was in quest for a %s "->"treasure"
)
def pick (voc: String) : String = {
  vocabulary(voc)(Random.nextInt(vocabulary(voc).length))
}

def fill_the_role (sentence: String, role : String) : String = {
  sentence.format(pick(role))
}

def tell_a_story {
   templates.foreach { case (sentence,role) =>
      print(fill_the_role(sentence,role))
   }
}
} // end class Story_Teller

val a_story_teller = new Story_Teller
a_story_teller.tell_a_story
```

Kojo

- This version is a little bit complex, since it uses class, data structures (Map type) for templates and roles and a foreach/case to navigate in them
- It show a data driven approach, that makes it simpler to enrich the code with new template and new characters
- In this version, there is no control about the persistence of the binding role/character through the program
- There is also a limitation in the number of variables per sentences, due to the implementation through the "old Java style" interpolation mechanism
- Upgrade: make it work!



```
import scala.util.Random

class Story_Teller {

val vocabulary = Map (
  "hero"->Array(" knight "," dwarf "," singer "),
  "kingdom"->Array("Xanadu "," Eurovision "," Atlantis"),
  "treasure"->Array("crown "," unique ring "," cup of icecream ")
)

val  templates = Map (
  "There was once upon a time a %s, "->"hero",
  "striving for %s "->"kingdom",
  ", that was in quest for a %s "->"treasure"
)

def pick (voc: String) : String = {
  val a_value = vocabulary(voc)(Random.nextInt(vocabulary(voc).length))
  return a_value
}

def fill_the_role (sentence: String, role : String) : String = {
  val item = pick(role)
  sentence.format(item)
}

def tell_a_story {
   templates.foreach { case (sentence,role) =>
      print(fill_the_role(sentence,role))
   }
}
} // end class Story_Teller

val a_story_teller = new Story_Teller
a_story_teller.tell_a_story
```

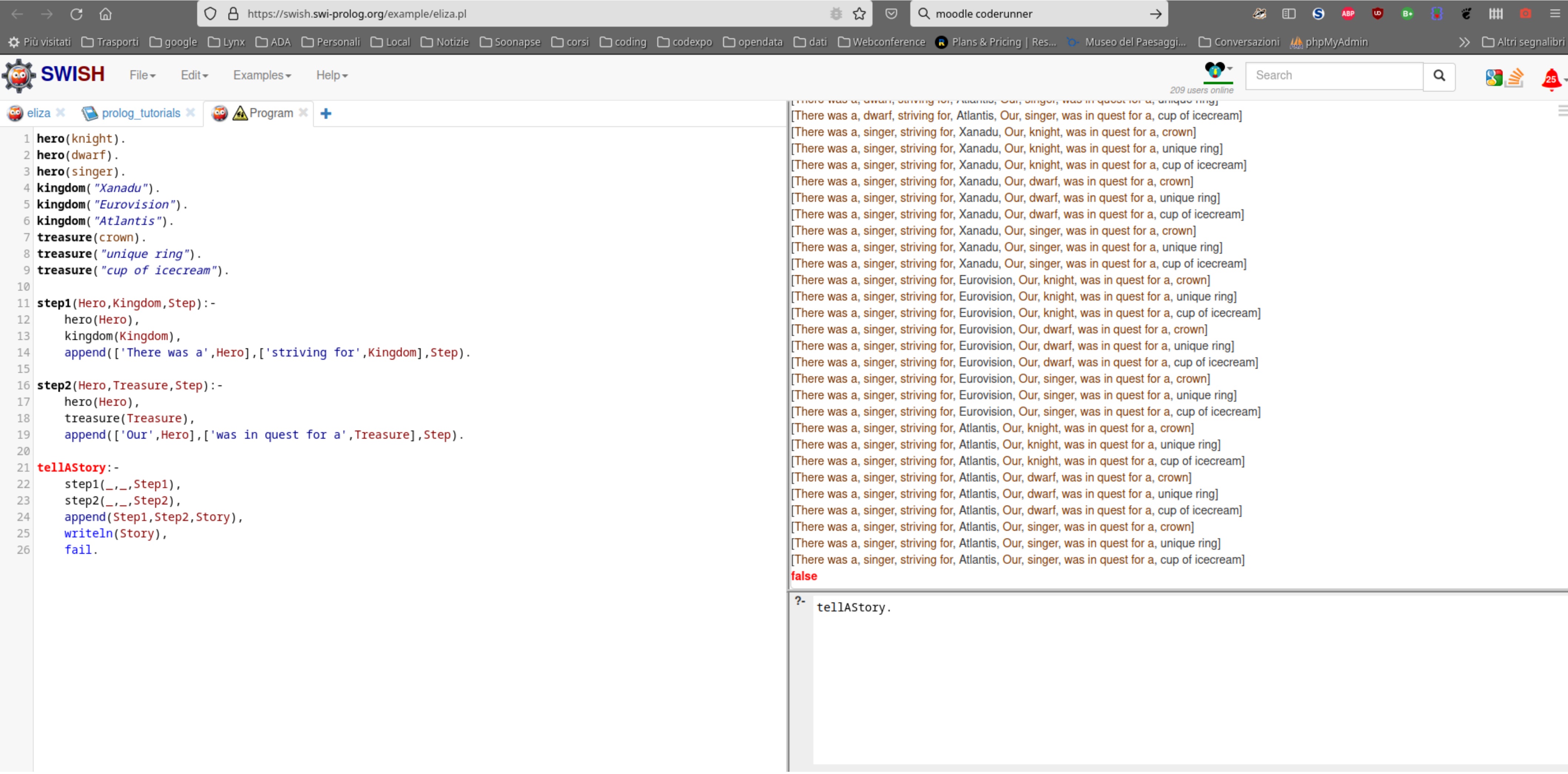
```
There was once upon a time a  knight , striving for  Atlantis , that was in quest for a
cup of icecream  ---
There was once upon a time a  knight , striving for Xanadu  , that was in quest for a
crown  ---
There was once upon a time a  singer , striving for  Atlantis , that was in quest for a
cup of icecream  ---
There was once upon a time a  knight , striving for  Eurovision  , that was in quest for a
cup of icecream
```

```
hero(knight).
hero(dwarf).
hero(singer).
kingdom("Xanadu").
kingdom("Eurovision").
kingdom("Atlantis").
treasure(crown).
treasure("unique ring").
treasure("cup of icecream").
```

```
step1(Hero,Kingdom,Step):-
     hero(Hero),
     kingdom(Kingdom),
    append(['There was a',Hero],['striving for',Kingdom],Step).

step2(Hero,Treasure,Step):-
    hero(Hero),
    treasure(Treasure),
    append(['Our',Hero],['was in quest for a',Treasure],Step).

tellAStory:-
     step1(_,_,Step1),
     step2(_,_,Step2),
     append(Step1,Step2,Story),
    writeln(Story),
    fail.
```

Prolog

- The third one has a typical Prolog structure with all facts stated at the beginning and a rule for every sentence. Thank to the unification mechanism, it is not necessary to bother about the bindings of roles.
- The main predicate finds all the stories and prints them, thanks to *fail* predicate.
- Upgrade: let the predicate randomly choose a single story using ! (cut)



```prolog
hero(knight).
hero(dwarf).
hero(singer).
kingdom("Xanadu").
kingdom("Eurovision").
kingdom("Atlantis").
treasure(crown).
treasure("unique ring").
treasure("cup of icecream").

step1(Hero,Kingdom,Step):-
    hero(Hero),
    kingdom(Kingdom),
    append(['There was a',Hero],['striving for',Kingdom],Step).

step2(Hero,Treasure,Step):-
    hero(Hero),
    treasure(Treasure),
    append(['Our',Hero],['was in quest for a',Treasure],Step).

tellAStory:-
    step1(_,_,Step1),
    step2(_,_,Step2),
    append(Step1,Step2,Story),
    writeln(Story),
    fail.
```

```
[There was a, dwarf, striving for, Atlantis, Our, singer, was in quest for a, unique ring]
[There was a, dwarf, striving for, Atlantis, Our, singer, was in quest for a, cup of icecream]
[There was a, singer, striving for, Xanadu, Our, knight, was in quest for a, crown]
[There was a, singer, striving for, Xanadu, Our, knight, was in quest for a, unique ring]
[There was a, singer, striving for, Xanadu, Our, knight, was in quest for a, cup of icecream]
[There was a, singer, striving for, Xanadu, Our, dwarf, was in quest for a, crown]
[There was a, singer, striving for, Xanadu, Our, dwarf, was in quest for a, unique ring]
[There was a, singer, striving for, Xanadu, Our, dwarf, was in quest for a, cup of icecream]
[There was a, singer, striving for, Xanadu, Our, singer, was in quest for a, crown]
[There was a, singer, striving for, Xanadu, Our, singer, was in quest for a, unique ring]
[There was a, singer, striving for, Xanadu, Our, singer, was in quest for a, cup of icecream]
[There was a, singer, striving for, Eurovision, Our, knight, was in quest for a, crown]
[There was a, singer, striving for, Eurovision, Our, knight, was in quest for a, unique ring]
[There was a, singer, striving for, Eurovision, Our, knight, was in quest for a, cup of icecream]
[There was a, singer, striving for, Eurovision, Our, dwarf, was in quest for a, crown]
[There was a, singer, striving for, Eurovision, Our, dwarf, was in quest for a, unique ring]
[There was a, singer, striving for, Eurovision, Our, dwarf, was in quest for a, cup of icecream]
[There was a, singer, striving for, Eurovision, Our, singer, was in quest for a, crown]
[There was a, singer, striving for, Eurovision, Our, singer, was in quest for a, unique ring]
[There was a, singer, striving for, Eurovision, Our, singer, was in quest for a, cup of icecream]
[There was a, singer, striving for, Atlantis, Our, knight, was in quest for a, crown]
[There was a, singer, striving for, Atlantis, Our, knight, was in quest for a, unique ring]
[There was a, singer, striving for, Atlantis, Our, knight, was in quest for a, cup of icecream]
[There was a, singer, striving for, Atlantis, Our, dwarf, was in quest for a, crown]
[There was a, singer, striving for, Atlantis, Our, dwarf, was in quest for a, unique ring]
[There was a, singer, striving for, Atlantis, Our, dwarf, was in quest for a, cup of icecream]
[There was a, singer, striving for, Atlantis, Our, singer, was in quest for a, crown]
[There was a, singer, striving for, Atlantis, Our, singer, was in quest for a, unique ring]
[There was a, singer, striving for, Atlantis, Our, singer, was in quest for a, cup of icecream]
false
```

```
?- tellAStory.
```

Part III

Comparison/1

- Which parameters can be used *by the teacher* to compare the three versions?
  - adequacy to the project goals
  - readability
  - conciseness
  - extensibility
  - ... ?

Comparison/2

- Which parameter can be used *by the students* to compare the three versions?
  - "this one was simpler to read/write" (= less words to type, less errors)
  - "this one seems a true program"
  - "this one was easy to debug"
  - … ?

Other versions

- Which language could be added to the list?
- Why?
- Could you think about a version in some visual language? Which one?
  - Which could be the advantages for the CS concepts learning?
  - Which could be other advantages?

Other activities

- How about changing the template and the word list to generate a different kind of text (a recipe, a wikipedia article, a poem, a love letter, an advertising, …)?
- How about generating automatically the *template*?
- Could we organize a "Turing game" among two teams, to see which one is able to write a program that write a text that seems human-written?