

Welcome to INFO216:  
Knowledge Graphs  
Spring 2023

Andreas L Opdahl  
<Andreas.Opdahl@uib.no>

# Session 8: Ontologies (OWL)

- Themes:
  - what and why?
  - basic OWL constructs (“RDFS-Plus”):
  - more precise properties
  - sameness and difference
  - complex classes (→ more later)
  - Programming in RDFLib



# Readings

- Sources:
  - Allemang, Hendler, Gandon (2020):  
**Semantic Web for the Working Ontologist**, 3<sup>rd</sup> edition:  
chapter 9-10 (“RDFS Plus”, but chapters 8-9 in the 2<sup>nd</sup> ed.)
  - Blumauer & Nagy (2020):  
Knowledge Graph Cookbook – Recipes that Work:  
e.g., pages 105-109, 123-124, (*supplementary*)
- Resources in the wiki <<http://wiki.uib.no/info216>>:
  - OWL 2 Primer, sections 2-6:  
<http://www.w3.org/TR/owl-primer/>
    - **show:** Turtle
  - VOWL: Visual Notation for OWL Ontologies

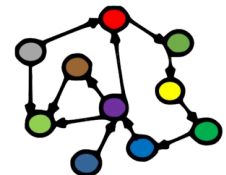


THE KNOWLEDGE GRAPH  
**COOKBOOK**  
RECIPES THAT WORK



ANDREAS BLUMAUER  
AND HELMUT NAGY

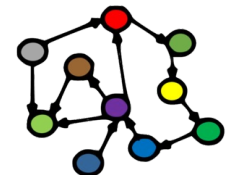
1st edition, 2020



# Web Ontology Language (OWL)

# Why do we need vocabularies?

- Shared, well-defined terms (dereferencable URIs) for types, properties and some individuals that can be used to represent a domain
- Domains can be:
  - people, their friends and workplaces (FOAF, BIO)
  - electronic and other documents (DC, BIBO)
  - commerce (schema.org)
  - classification in libraries etc. (SKOS)
  - general encyclopedic information (DBpedia, Wikidata)
  - general time and place (OWL-Time, geo)
  - ...and *lots* of others (→S09)



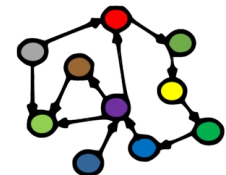
# Why do we need vocabularies?

- To make knowledge graphs more precisely defined
- To make semantic data sets easier to use
  - encourage reuse
  - avoid misunderstandings and errors
  - easier to understand, recombine, enrich...
- To support computer processing
  - more powerful
  - more general



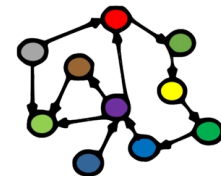
# RDFS is a useful starting point...

- We can say:
  - “a president is a politician” so that
    - saying “Trump is a president” entails saying “Trump is a politician”
  - “a politician is a human” so that
    - saying “Trump is a president” *also* entails saying “Trump is a human”
  - “the president of something is a politician” so that
    - saying “Trump is a president of U.S.A.” entails “saying Trump is a politician”
  - “something having a president is a country” so that
    - saying “Trump is a president of U.S.A.” entails saying “U.S.A. is a country”
  - “being president also means being citizen” so that
    - saying “Trump is a president of U.S.A.” entails saying “Trump is a citizen of U.S.A.”
- *RDFS expresses this but not (so much) more...*



# RDFS is a useful starting point...

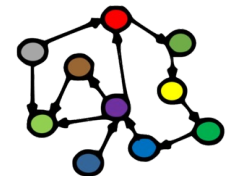
- But lots of simple stuff it cannot express, e.g.:
  - “every ancestor of an ancestor is an ancestor too”
  - “the BirthNumber of a Person is unique”
  - “a Republic has exactly one President”
  - “a FootballTeam has 11 activePlayers, a VolleyballTeam 6”
  - “a StringQuartet has two violins but only one viola and one cello”
  - “classes with different URIs actually represent the same class”
  - “resources with different URIs represent the same resource”
  - “properties with different URIs are actually the same”
  - “two individuals are different”, “two classes are disjoint”
  - “a class is a union (or intersection) of other classes”
  - “a class is a negation of another class”
- *OWL expresses all this and more!*

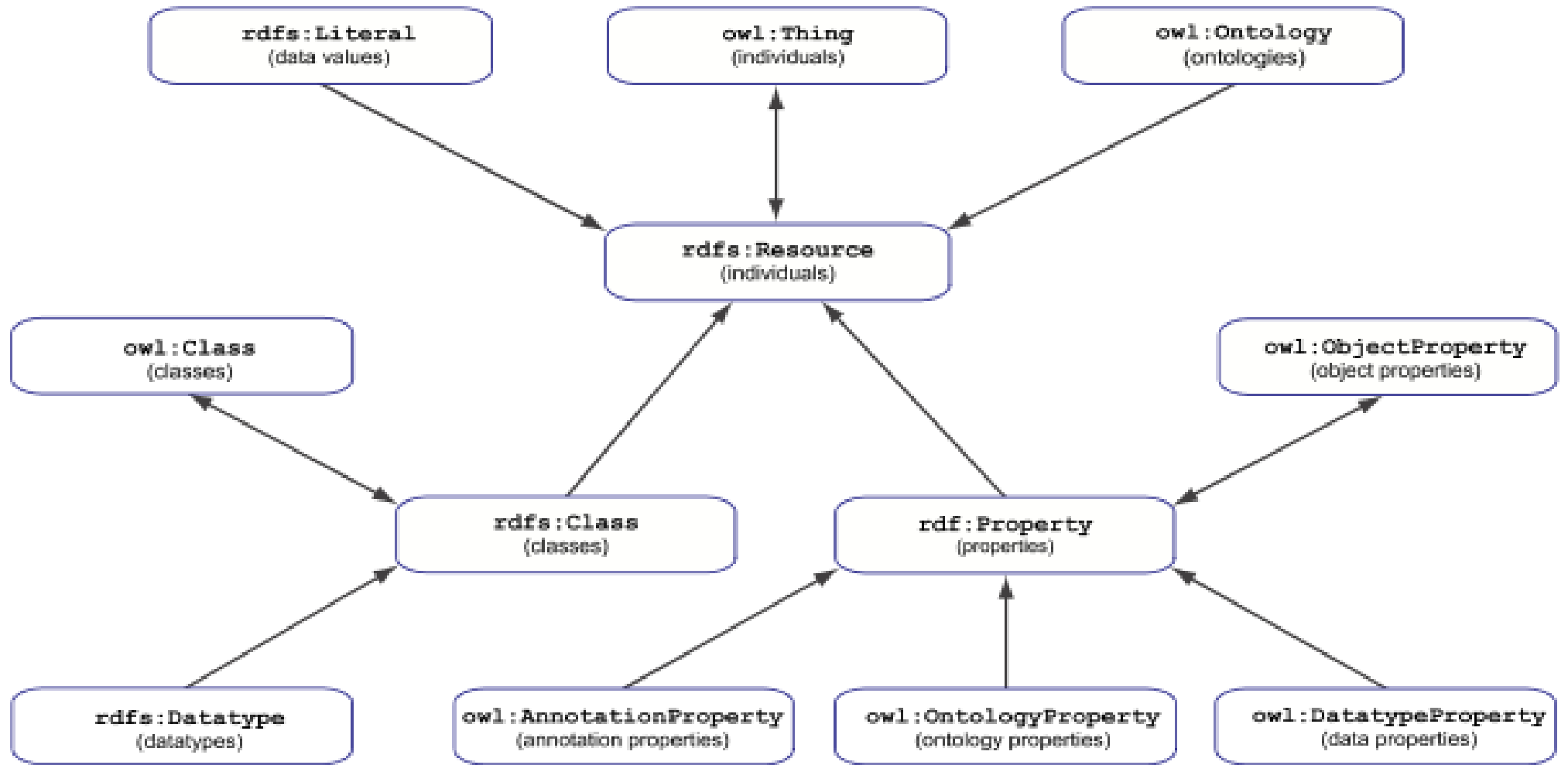




# Basic idea

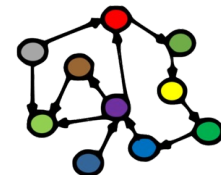
- Web Ontology Language (OWL):
  - builds on RDF and RDFS
  - uses classes and properties from RDF and RDFS
  - adds precision and formality
- Basic OWL-concepts:
  - `owl:Thing` (equivalent to `rdfs:Resource`)
  - `owl:Class` (equivalent to `rdfs:Class`)
  - `owl:ObjectProperty` (equivalent to `rdf:Property`)
- Good practice: keep *Classes*, *Individuals* (that are not classes), and *Properties* disjoint, i.e., no resource has more than one of them as *rdf:type*





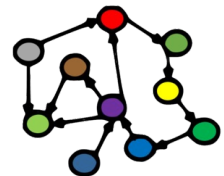
# What does OWL offer?

- Extensions of RDFS, e.g.:
  - more *specific types* of properties
  - *identical and different* classes, properties, individuals
  - *defining new classes*:
    - complex classes (union, intersection, complement)
    - property restrictions, enumeration of individuals
  - *defining new properties* based on existing ones
  - *mathematical formality* (for large parts of OWL)
    - (more on this later)



# Reuses or specialises RDFS

- *Reused* in OWL:
  - `rdf:type`, `rdf:Property`,  
`rdfs:subClassOf`, `rdfs:subPropertyOf`, `rdfs:domain`, `rdfs:range`
  - ...and lots of other stuff...
- *Renamed* by OWL:
  - `owl:Thing` (equivalent to `rdfs:Resource`)
  - `owl:Class` (equivalent to `rdfs:Class`)
  - `owl:ObjectProperty` (equivalent to `rdf:Property`)
- *Specialised* by OWL:
  - everything else in OWL *specialises* something in RDF / RDFS
  - but also introduces its own, and more powerful, formal underpinning

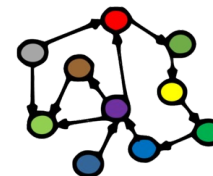


# Basic OWL ("RDFS Plus")

# Inverse properties

- Properties can be inverses (or reverses) of one another, e.g.,
  - `ex:DonaldTrump ex:presidentOf ex:USA .`
  - `ex:USA ex:hasPresident ex:DonaldTrump .`
- `P1 owl:inverseOf P2`:
  - `ex:presidentOf owl:inverseOf ex:hasPresident .`
  - `owl:inverseOf owl:inverseOf owl:inverseOf .`
  - `owl:inverseOf` a `owl:ObjectProperty` .
- Entailment rules:
  - if *`P1 owl:inverseOf P2`* then
    - *`P2 owl:inverseOf P1 .`*
  - if *`S P1 O . P1 owl:inverseOf P2`* then
    - *`O P2 S .`*

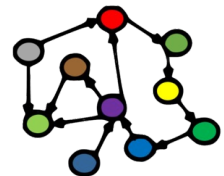
*The axioms  
are informative  
(not mandatory)*



# Symmetric properties

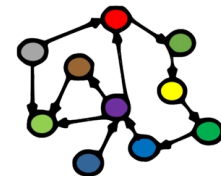
- Some properties are their own inverse, e.g.,
  - `ex:DonaldTrump ex:marriedTo ex:MelianaTrump .`
  - `ex:MelianaTrump ex:marriedTo ex:DonaldTrump .`
- `P rdf:type owl:SymmetricProperty:`
  - `ex:marriedTo a owl:SymmetricProperty .`
  - `owl:inverseOf a owl:SymmetricProperty .`
  - `owl:SymmetricProperty rdfs:subClassOf owl:ObjectProperty .`
- Entailment rules:
  - if  *$P$  a `owl:SymmetricProperty`* then
    - *$P$  `owl:inverseOf`  $P$  .*
  - if  *$S P O$  .  $P$  a `owl:SymmetricProperty`* then
    - *$O P S$  .*

*The axioms  
are informative  
(not mandatory)*



# Asymmetric, reflexive, irreflexive properties

- New in OWL2:
  - both *symmetric* and *asymmetric* properties:
    - ex:marriedTo a owl:SymmetricProperty .
      - “marriage is always mutual (two-way)”
    - ex:hasChild a owl:AsymmetricProperty .
      - “two resources cannot be the child of each other”
    - *many properties are neither – leave it open!*
  - both *reflexive* and *irreflexive* properties:
    - owl:sameAs a owl:ReflexiveProperty .
      - “every resource is the same as itself”
    - ex:hasChild a owl:IrreflexiveProperty .
      - “no resource can be its own child”
    - *many properties are neither – leave it open!*

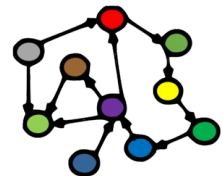




# Transitive properties

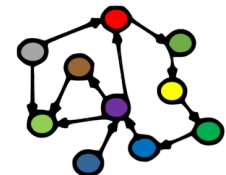
- Some properties can form chains so that the result is the property itself, e.g.:
  - `ex:DonaldTrump ex:hasPredecessor ex:BarackObama .`
  - `ex:BarackObama ex:hasPredecessor ex:GeorgeWBush .`
  - `ex:DonaldTrump ex:hasPredecessor ex:GeorgeWBush .`
- `P a owl:TransitiveProperty`:
  - `ex:hasPredecessor a owl:TransitiveProperty .`
  - `rdfs:subClassOf a owl:TransitiveProperty .`
  - `rdfs:subPropertyOf a owl:TransitiveProperty .`
- Entailment rules:
  - “if  $S P X . X P O . P a owl:TransitiveProperty$  then
    - $S P O .$ ”

*The axioms  
are informative  
(not mandatory)*



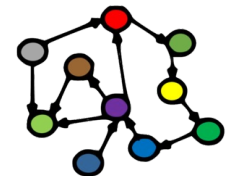
# Functional properties

- Each subject *can only have one* object value for the functional property, e.g.,
  - `ex:hasPresident` a `owl:FunctionalProperty` .
  - `ex:dateOfBirth` a `owl:FunctionalProperty` .
  - `owl:FunctionalProperty` `rdfs:subClassOf owl:ObjectProperty` .
- Entailment rule:
  - if  $S P O1 . S P O2 . P$  a *owl:FunctionalProperty* then
    - $O1$  *owl:sameAs*  $O2$  .
  - The rule also holds for *owl:DatatypeProperties*, but:
    - if two different literals become asserted as *owl:sameAs* one another, *the ontology is inconsistent*



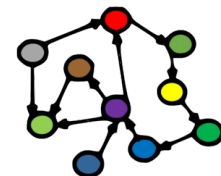
# Inverse functional properties

- Two different subjects cannot have the same object for an inverse functional property, i.e.,
  - `ex:presidentOf` a `owl:InverseFunctionalProperty` .
- Inverse functional properties are *unique* for each individual
  - used for *identifiers* in OWL 1
  - OWL 2 has a built-in *owl:hasKey* property for identifiers:
    - similar to inverse functional object properties
    - can only be used with OWL 2's *owl:NamedIndividuals*
    - ...not for anonymous “*owl:Individuals*”



# Summary: more precise properties

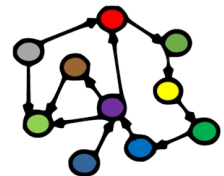
- `owl:inverseOf`
- `owl:SymmetricProperty`, `owl:AsymmetricProperty`
- `owl:ReflexiveProperty`, `owl:IrreflexiveProperty`
- `owl:TransitiveProperty`
- `owl:FunctionalProperty`, `owl:InverseFunctionalProperty`
- `owl:hasKey`
- Also:
  - negated properties (later)
  - chained properties, e.g.:  
`fam:hasGrandmother`  
`owl:propertyChainAxiom`  
`( :hasParent :hasMother ) .`



# Individual equivalence

- Sometimes, two individuals (with different URI-s) represent the same thing:
  - [http://dbpedia.org/resource/Donald\\_Trump](http://dbpedia.org/resource/Donald_Trump)
  - <http://wikidata.org/entity/Q22686>
- I1 owl:sameAs I2:
  - owl:sameAs a owl:ReflexiveProperty .
  - owl:sameAs a owl:SymmetricProperty .
  - owl:sameAs a owl:TransitiveProperty .
- owl:sameAs is an *equivalence relation*:
  - because it is *reflexive*, *symmetric* and *transitive*

The axioms  
are informative  
(not mandatory)



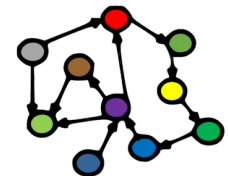
# Unique Name Assumption (UNA)

- If two resources have different names, do they necessarily represent different things?
- RDF and OWL does *not* assume this!
  - *in RDF and OWL, we do not know whether resources with different names represent different things or not*
- We can use
  - `owl:sameAs` – two resources represent the same thing!
  - `owl:differentFrom` – they represent different things!
- Some ICT-languages and technologies use UNA, others do not!



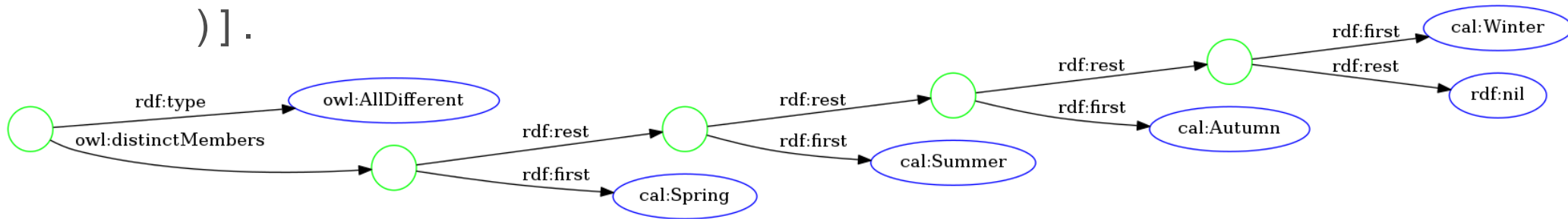
# Individual difference

- Sometimes, a *pair* of individuals with different names (URI-s) represent *different* things, e.g.,
  - cal:Spring owl:differentFrom cal:Summer .
- owl:differentFrom
  - *not* transitive
  - *not* reflexive



# Individual difference

- Sometimes, a pair of individuals with different names (URI-s) represent different things, e.g.,
  - cal:Spring owl:differentFrom cal:Summer .
- Sometimes, a *group* of individuals with different names (URI-s) *all* represent *different* things, e.g.,
  - [ a owl:AllDifferent ;  
owl:distinctMembers (  
cal:Spring cal:Summer cal:Autumn cal:Winter  
) ] .

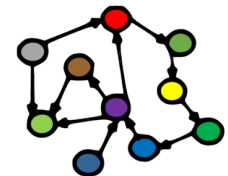


Namespaces:  
rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>  
cal: <http://ex.org/cal/>  
owl: <http://www.w3.org/2002/07/owl#>



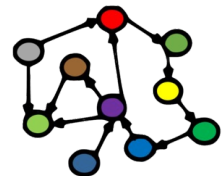
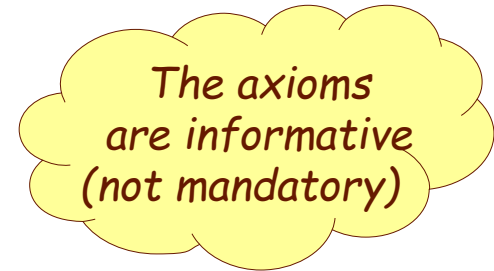
# Individual difference

- Sometimes, a pair of individuals with different names (URI-s) represent different things, e.g.,
  - `cal:Spring owl:differentFrom cal:Summer` .
- Sometimes, a group of individuals with different names (URI-s) all represent different things, e.g.,
  - [ `a owl:AllDifferent ; owl:distinctMembers ( cal:Spring cal:Summer cal:Autumn cal:Winter ) ]` .
  - *owl:AllDifferent* and *owl:distinctMembers* are special constructs in OWL
    - they must always be used together
  - ...corresponds to pairwise *owl:differentFrom* between *all* individuals in the *owl:distinctMembers*-list



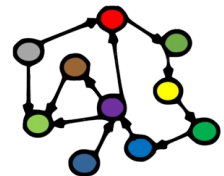
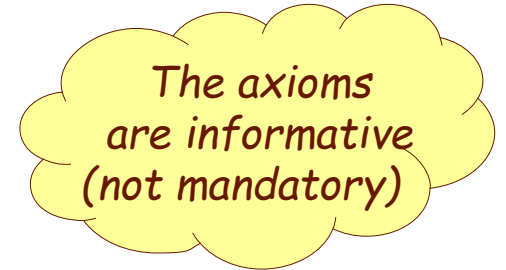
# Equivalent classes

- Sometimes, two classes (with different URI-s) represent the *same* class:
- C1 owl:equivalentClass C2:
  - owl:equivalentClass a owl:ReflexiveProperty .
  - owl:equivalentClass a owl:SymmetricProperty .
  - owl:equivalentClass a owl:TransitiveProperty .
- owl:equivalentClass is another *equivalence relation*:
  - it is *reflexive*, *symmetric* and *transitive*
- C1 owl:equivalentClass C2 means the same as
  - C1 rdfs:subClassOf C2 and C2 rdfs:subClassOf C1
- Also *disjoint* classes:
  - *uib:InternalCensor*  
*owl:disjointWith skos:ExternalCensor .*



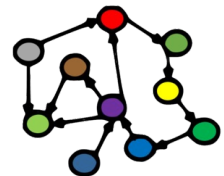
# Equivalent properties

- Two properties (with different URI-s) can represent the same property:
- P1 owl:equivalentProperty P2:
  - owl:equivalentProperty a owl:ReflexiveProperty .
  - owl:equivalentProperty a owl:SymmetricProperty .
  - owl:equivalentProperty a owl:TransitiveProperty .
- owl:equivalentProperty is another *equivalence relation*:
  - it is *reflexive*, *symmetric* and *transitive*
- Also *disjoint* properties:
  - skos:prefLabel owl:propertyDisjointWith skos:altLabel .



# Summary: sameness and difference

- Individuals:
  - pairwise: `owl:sameAs`, `owl:differentFrom`
  - groupwise difference: `owl:AllDifferent`
- Classes:
  - pairwise: `owl:equivalentClass`, `owl:disjointWith`
  - groupwise difference: `owl:AllDisjointClasses`
- Properties:
  - pairwise: `equivalentProperty`, `propertyDisjointWith`
  - groupwise difference: `owl:AllDisjointProperties`
- Membership in the groups:
  - `owl:distinctMembers` (*preferred*) or `owl:members`



# Basic OWL reasoning in Python and rdflib

# Basic OWL inference in RDFLib

- `import owlrl`

...

```
DeductiveClosure(RDFS_Semantics).expand(graph) # RDFS reasoning
```

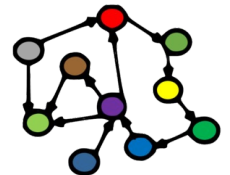
...

```
DeductiveClosure(OWLRL_Semantics).expand(graph) # OWL-RL reasoning
```

...

```
DeductiveClosure(OWLRL_Extension,  
                 rdfs_closure = True, axiomatic_triples = True,  
                 datatype_axioms = True).expand(graph) # Maximum reasoning
```

...

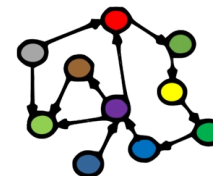


# Complex OWL classes

# Union classes

- A union class contains all the individuals in *either of two* or more other classes, e.g.,
  - foaf:Agent  
a owl:Class;  
owl:unionOf ( foaf:Person foaf:Organization ) .
- Entailment rule:
  - if *C owl:equivalentClass [ owl:unionOf ( C1... Cn ) ]* then
    - *C1 rdfs:subClassOf C . ... Cn rdfs:subClassOf C .*
- why not say just, e.g.,:
  - foaf:Person rdfs:subClassOf foaf:Agent .
  - foaf:Organization rdfs:subClassOf foaf:Agent .

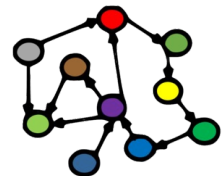
Actually, FOAF defines more types of Agents than this!





# Intersection classes

- An intersection class contains all the individuals in *all of* two or more other classes, e.g.
  - `uib:StudentAssistant`  
a `owl:Class`;  
`owl:intersectionOf ( uib:Student uib:Teacher ) .`
- Entailment rule:
  - if *C owl:equivalentClass [ owl:intersectionOf ( C1... Cn ) ]* then
    - *C rdfs:subClassOf C1 . ... C rdfs:subClassOf Cn .*
- why not say, e.g.:
  - `uib:StudentAssistant rdfs:subClassOf uib:Student .`
  - `uib:StudentAssistant rdfs:subClassOf uib:Teacher .`



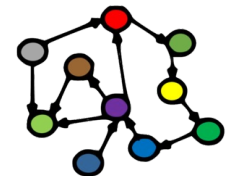
# Complement classes

- A complement class contains all the individuals *that are not* in another class:
  - `uib:ExternalCensor owl:complementOf uib:InternalCensor .`



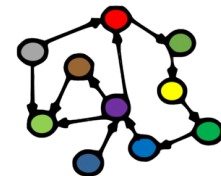
# Complement classes

- A complement class contains all the individuals *that are not* in another class:
  - `uib:ExternalCensor owl:complementOf uib:InternalCensor .`
  - *...but is this correct?!*



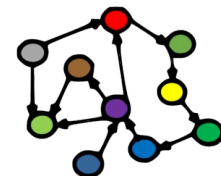
# Complement classes

- A complement class contains all the individuals *that are not* in another class:
  - `uib:ExternalCensor`  
    `a owl:Class;`  
    `owl:complementOf uib:InternalCensor .`



# Complement classes

- A complement class contains all the individuals *that are not* in another class:
  - `uib:ExternalCensor`  
`owl:intersectionOf (`  
    `uib:Censor`  
    `owl:complementOf uib:InternalCensor`  
`)`.



# Complement classes

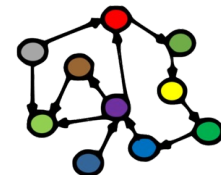
- A complement class contains all the individuals *that are not* in another class:
  - `uib:ExternalCensor`

```
owl:intersectionOf (  
  uib:Censor  
  [  
    a owl:Class ;  
    owl:complementOf uib:InternalCensor  
  ]  
).
```



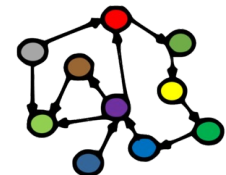
# Complement classes

- A complement class contains all the individuals *that are not* in another class:
  - `uib:ExternalCensor`  
`owl:intersectionOf (`  
    `uib:Censor`  
    `[ owl:complementOf uib:InternalCensor ]`  
`)`.



# Closed World Assumption (CWA)

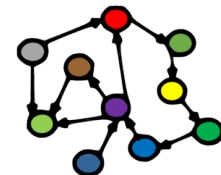
- Whenever something is not explicitly stated in the ontology, can we assume that the opposite is the case?
  - DBpedia only lists three James Dean movies – can we thus assume that he only played in three?
- Classical logic and many ICT languages assume so:
  - this is the “*Closed World Assumption*” (CWA)
- *In RDF and OWL, we do not assume that something is false just because it is not stated*
  - this is the “*Open World Assumption*” (OWA)





# Enumeration classes

- An *enumeration class* is defined by exhaustively listing all its member individuals, e.g.:
  - [ a owl:Class ;  
owl:oneOf ( cal:Spring ... cal:Winter ) ] .
- An enumeration class is *closed*
  - there are no other member individuals
  - ensured by using *RDF Collections*:
    - rdf:List, rdf:first, rdf:rest, rdf:nil
- **Does *not* imply** that the individuals are **distinct**
  - this must be stated explicitly



# Other ways to write complex classes

- Why can also write:

```
cal:Season
```

```
  owl:oneOf ( cal:Spring ... cal:Winter ) .
```

or

```
cal:Season owl:equivalentClass [
```

```
  owl:oneOf ( cal:Spring ... cal:Winter ) ] .
```

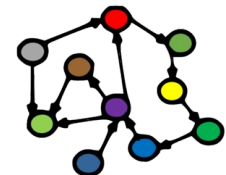
- or (a weaker claim):

```
cal:Season owl:subClassOf [
```

```
  owl:oneOf ( cal:Spring ... cal:Winter ) ] .
```

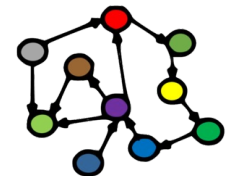
- Reason:

- *rdfs:subClassOf* can be computationally more efficient
- *owl:equivalentClass* is sometimes implemented as a costly two-way *rdfs:subClassOf*



# Summary: complex classes

- owl:oneOf
- owl:unionOf
- owl:intersectionOf
- owl:complementOf (and the CWA)
- owl:NegativePropertyAssertion, owl:sourceIndividual, owl:assertionProperty, owl:targetIndividual



**Next week:  
Vocabularies**