

## Datové Formáty

### 1. Přednáška

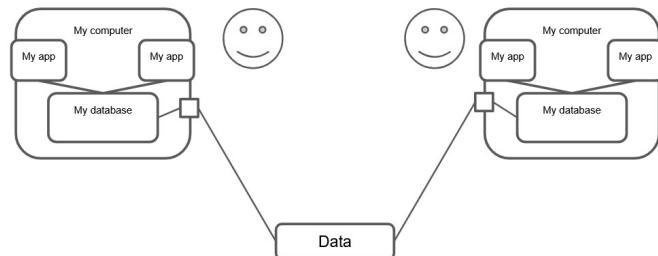
#### Základní pohled

→ je důležité se koukat na data jako taková, ne jen jako část aplikace

-- nechci dát data konkurenci, když si

koupí aplikaci

-- určím si tedy, co přesně posílám/sdílím z/do aplikace  
-- data nezávislá na aplikacích



→ co za formáty se celkově (na webu)

používají, převody

→ formáty, schémata

---- toto vše vede k větší přehlednosti, které chceme dosáhnout (+ ne-psaná pravidla)

#### Konceptuální pohled na data (Conceptual domain model)

→ přehled nezávislý na technologiích v tom, co za formáty bych v reálném světě potřeboval

→ „ne-IT“ oblast – detailly vazeb, vztahů (O čem jsou data?)

→ příklady:

UML:

**Class** (Třída): Catalog

**Attributes** (Atributy třídy): title, description --- každá instance má dané atributy  
- vztahy → **asociace** („je částí“ + multiplicita)

#### Datový model

→ logická reprezentace dat (sada nástrojů)

- typy:

- grafy** (RDF, LPG, ...) --- uzly a hrany
- hierarchie** (DOM, JSON, ...)
- tabulky** (relační model, ...)

#### Datový formát

→ fyzický pohled na data a jak budou zapsána a serializovaná

grafové:

**RDF** – serializace do *textových formátů* (N-Triples, N-Quads, **Turtle**, ...)

**LPG** – do **CSV** či **JSONu**

hierarchické:

**DOM** – do **XML**, **HTML**

**JSON** – do **JSON**, **XML**

relační:

→ do **CSV**, **SQL dump**

#### Datové schéma

→ různé způsoby vyjádření v daném formátu --- chceme tedy nějak **validovat**

- **anotace** a **omezení aplikovatelná** na **instance dat** (popis + validace)

- příklady: **JSON schema**, **CSV on web**, **XML schema**

- **meta-formát** = hostitelský formát (úprava na specifický formát přes schéma)

--- GeoJSON, SVG, ...

## Datový typ

- „low-level“
- stejné napříč datovými formáty (např. bool, int, date, ...)

### Vlastnosti datových formátů

#### - otevřený

→ specifikace daného formátu je volně dostupná a volně využitelná

#### - zavřený

→ potřeba certifikace na to, abych daný d.f. mohl používat

#### - strojová čitelnost (machine-readable)

- jsou data jednoduše zpracovatelná nějakou aplikací?
- jasná a čitelná struktura, jednotlivé znaky/části přístupné

#### - binární

- struktura po bitech
- není to textový soubor!
- hex editorem jde prohlížet

#### - textový

- obsahuje text (jde taktéž prohlížet hex editorem)
- typicky strukturován po rádcích
- text je encoded do 0 a 1 skrze **character encoding** (US-ASCII – 7 bitů per znak)
- **UTF-8** = jeden znak 1 až 4 bajty
- konce řádků: CR / LF
- BOM --- 3 bajty, ukazuje použití nějakého kódování

## Standardizace (Standardy)

- specifikace určující nějaká pravidla --- **interoperabilita**
- příklady:

IETF: vymýšlý standardy

W3C: mezinárodní standardy pro www

RFC 2119 → jak psát specifikace, co za klíčová slova používat  
(MUST, REQUIRED, SHALL; MUST NOT, SHALL NOT, ...)

## Identifikátor (konkrétní datové entity)

- když najdu **dva stejné identifikátory = stejná věc** (v opačném případě ne)
- příklady:

URI – Uniform Resource **Identifier** (**jen id bez funkce najít**)

URN – Uniform Resource **Name** (typ URI, lokalizaci dané věci neřeší)

URL – Uniform Resource **Locator** (vždy funguje v prohlížeči, **najde zdroj**)

IRI – dovoluje používat utf-8 (to URI ani URL ne)

- boolean
  - = true
  - = false
- number
  - = integer
    - = 42
  - = decimal
    - = 42.42
  - = float/double
    - = 4.2e2
- date - **ISO-8601-compliant**
  - = YYYY-MM-DD
  - = 2021-03-01
- time
  - = HH:MM:SS.sss
  - = 18:40:00
- dateTime
  - = YYYY-MM-DDTHH:MM:SS.sss
  - = 2021-03-01T18:40:00
  - = 2021-03-01T18:40:00+02:00
- time zones
  - = 2021-03-01T18:40:00+02:00
  - = 2021-03-01T18:40:00
  - = 2021-03-01Z

## 2. Přednáška (GRAFOVÉ DATOVÉ FORMÁTY)

-- vztahy nahrazuji (např. „has description“) pomocí URL!

- **prefixování** (např. *ex: description*)

### RDF (Orientovaný multigraf)

- set trojic: **subjekt, vlastnost, objekt**

- literál hodnota (např. „education“)

- nemá uspořádání

- **prázdné uzly** (blank nodes) bez IRI

- serializace RDF

a) **N-Triples** (langTag či IRI jako objekt)

b) **RDF Turtle** (prefixy, zjednodušení pro stejné subjekty)

--- relativní IRI @base

--- rdf:type = a

--- pojmenované grafy (Quads)

- **RDF dataset**

→ množina pojmenovaných grafů a jednoho výchozího grafu (TriG)

- **RDF schéma**

→ hierarchie, podmnožiny

→ **vlastní slovníky, typy**

- **RDFS --- všechny platí najednou**

--- domain, range, property

--- label, comment, seeAlso,...

- chci zachovat pořadí RDF = **rdf>List**

--- jako spojový seznam

--- v Turtlu ()

- máme i podobné jako Bag, Seq, Alt

- OWA = **Open World Assumption**

--- když nemám záznam - „Unknown“, neříkáme „Ne“ (to je Closed world)

- **linked data**

--- když je nemám, tak se můžu zaseknout (zájemné identifikátory, složitý přístup, kontext částečně či zcela chybí)

--- nejlépe machine-readable a srozumitelné

--- **vše identifikovat pomocí URI**

--- vždy radši http **URI se standardy** (rdf, sparql)

--- **přidání i relevantních URIs** (přidává hodně co se týče tohoto principu)

--- mířím do API aplikace, to však nemusí pomoci

There is a thing, which is a catalog

The catalog has title "my catalog"

ex:catalog rdf:type dcat:Catalog .

ex:catalog dcterms:title "my catalog" .

ex:catalog dcterms:description "my first testing catalog" .

ex:catalog foaf:homepage ex:homepage .

ex:homepage rdf:type foaf:Page .

The catalog has description

"my first testing catalog"

The catalog has homepage

"https://mycatalog.example.org/homepage"

<http://purl.org/dc/terms/creator>

=

@prefix dcterms: <http://purl.org/dc/terms/> .  
dcterms:creator

<http://one.example/subject1> <http://one.example/predicate1>  
<http://one.example/object1> . # comments here

<http://example.org/show/218> <http://example.org/show/localName> "That  
Seventies Show"@en . # literal with a language tag

<http://en.wikipedia.org/wiki/Helium>  
<http://example.org/elements/atomicNumber>  
"2"^^<http://www.w3.org/2001/XMLSchema#integer> . # xsd:integer

### RDF 1.1 Turtle - Prefixes and ";" and ","

<http://example.com/index.html> <http://purl.org/dc/terms/created> "2020-04-23"^^<http://..#date> .  
<http://example.com/index.html> <http://purl.org/dc/terms/creator> <http://example.com/staff/8574> .  
<http://example.com/index.html> <http://purl.org/dc/terms/creator> <http://example.com/staff/8575> .  
<http://example.com/index.html> <http://purl.org/dc/terms/title> "Moje stránka"@cs .  
<http://example.com/index.html> <http://purl.org/dc/terms/title> "My page"@en .

@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .  
@prefix dcterms: <http://purl.org/dc/terms/> .  
@prefix my: <http://example.com/> .  
@prefix staff: <http://example.com/staff/> .  
  
my:index.html dcterms:created "2020-04-23"^^xsd:date ;  
dcterms:creator staff:8574 ,  
staff:8575 ;  
dcterms:title "Moje stránka"@cs ,  
"My page"@en .

### RDF 1.1 Turtle - playing with prefixes and rel. IRIs

# In-scope base IRI is the document URI at this point  
<a1> <b1> <c1> .  
@base <http://example.org/ns/> .

# In-scope base IRI is http://example.org/ns/ at this point  
<a2> <http://example.org/ns/b2> <c2> .  
@base <foo/> .

# In-scope base IRI is http://example.org/ns/foo/ at this point  
<a3> <b3> <c3> .  
@prefix : <bar#> .  
:a4 :b4 :c4 .  
@prefix : <http://example.org/ns2#> .  
:a5 :b5 :c5 .

## RDF - relation to conceptual model

### 3. Přednáška

- RDF: relační do conceptuálního

### SPARQL

- dotazovací jazyk pro RDF

- Sparql endpoint – http služba, která přijímá sparql queries a posílá je

- syntax: proměnná „?name“

- důležité použití **optional**  
(jiné věci musí být splněny)

- UNION obsahuje duplicitu

- DISTINCT

- FILTER – omezení pro řádky (age>27)

- LIMIT – počet výsledků omezení (10 je standard u věcí, co neznám)

- SERVICE – „federated query“

- BIND – např. cena po slevě

BIND (?p\*(1-?discount) AS ?price)

např.:

PREFIX sis: <http://..../sis#>

SELECT ?name ?page

WHERE {

?stud a ?type ;

sis:name ?name ;

sis:age ?age .

}

**Blank nodes** – mají vždy random ID

Literály – jazykové tagy @cs, int, ...

+ fungují standardní operátory

Podporuje **vnořené dotazy!**

(Select v Selectu)

- **VALUES** ?book { :book1 }

--- musí namatchovat přímo book1

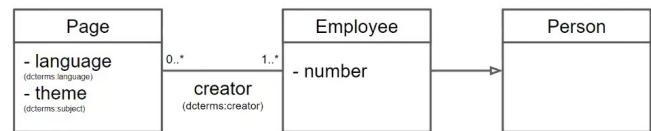
- **ASK** (odpověď je true/false)

- **DESCRIBE** – vrátí rdf graf o subjektu/objektu

- **ORDER BY** – možnost srovnání dle para

- **CONSTRUCT** → vytvoříme „nový graf“, protože např. z 2 částí jména udělám jednu

```
CONSTRUCT
{ ?s sis:name ?fullName }
WHERE {
  ?s sis:firstName ?n1 ;
      sis:lastName ?n2 .
  BIND(CONCAT(?n1, " ", ?n2) AS ?fullName)
}
```



```

ex:index.html a ex:Page .
ex:index.html dcterms:creator emp:85740 .
ex:index.html dcterms:subject "education" .
ex:index.html dcterms:language "en" .

emp:85740 a ex:Employee .
emp:85740 ex:number 42 .

ex:Page a rdfs:Class .
ex:Employee a rdfs:Class .
ex:Person a rdfs:Class .
ex:Employee rdfs:subClassOf ex:Person .

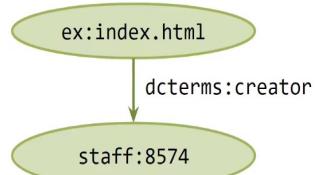
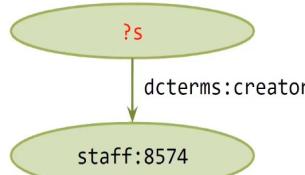
dcterms:creator a rdf:Property .
dcterms:subject a rdf:Property .
dcterms:language a rdf:Property .
ex:number a rdf:Property .
  
```

?s

dcterms:creator  
staff:8574 .

ex:index.html  
dcterms:creator  
staff:8574 .

?s <- ex:index.html



```

sis:stud1 sis:name "John" ;
           sis:age 26 ;
           rdf:type sis:Person .

sis:stud2 sis:name "Peter" ;
           sis:age 30 ;
           rdf:type sis:Person .

sis:stud3 sis:name "Martin" ;
           sis:age 20 .
  
```

```

?stud sis:name ?name ;
      sis:age ?age .
OPTIONAL {
  ?stud rdf:type ?type .
}
  
```

?stud	?name	?age	?type
sis:stud1	"John"	26	sis:Person
sis:stud2	"Peter"	30	sis:Person
sis:stud3	"Martin"	20	NOT BOUND

Solutions table

SPARQL querying - union graph pattern matching

```

sis:stud1 sis:name "John" ;
           sis:age 26 ;
           rdf:type sis:Person .
  
```

```

sis:stud2 sis:name "Peter" ;
           sis:age 30 ;
           rdf:type sis:Person .
  
```

```

sis:stud3 sis:name "Martin" ;
           sis:age 20 .
  
```

```

{
  ?stud sis:name ?name ;
        sis:age ?age .
}
UNION
{
  ?stud sis:name ?name ;
        sis:age ?age ;
        rdf:type ?type .
}
  
```

?stud	?name	?age	?type
sis:stud1	"John"	26	sis:Person
sis:stud2	"Peter"	30	sis:Person
sis:stud3	"Martin"	20	NOT BOUND
sis:stud1	"John"	26	NOT BOUND
sis:stud2	"Peter"	30	NOT BOUND

```

PREFIX schema: <http://schema.org/>
PREFIX gr: <http://purl.org/goodrelations/v1#>
  
```

```

SELECT (MAX (?fine) AS ?max)
WHERE
{
  ?check a schema:CheckAction ;
         schema:result/schema:result/gr:hasCurrencyValue ?fine .
}
  
```

#### 4. Přednáška

#### RDF slovníky

- standardizace

→ jméno / label / title ???

#### Dublin Core

- základní vlastnosti pro popis knih → *title, publisher, date*

- dc a dcterms

#### SKOS (prefix skos)

- Simple Knowledge Organization System

- skos:Concept (core class = nápad, notace)

- skos:ConceptScheme (sjednocení více konceptů)

- skos:inScheme (v jakém schématu je)

- skos:prefLabel (preferovaný název, 1 pro jazyk) + altLabel, hiddenLabel (misspellings – jde najít i přes chyby uživatele)

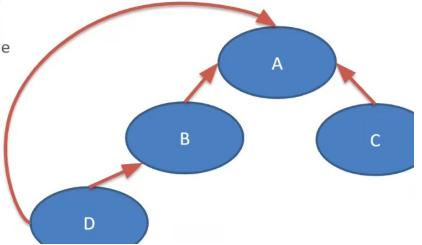
- skos:notation

- skos:Collection (skos:member / memberList)

```
<MyNestedCollection> a skos:Collection ;
  skos:member <X> , <Y> , <Z> .
```

```
<MyConcept>
  skos:prefLabel "animals"@en ;
  skos:altLabel "fauna"@en ;
  skos:hiddenLabel "aminalis"@en ;
  skos:prefLabel "animaux"@fr ;
  skos:altLabel "faune"@fr .
```

- skos:semanticRelation
  - skos:related
  - skos:broaderTransitive
    - skos:broader
  - skos:narrowerTransitive
    - skos:narrower



#### GoodRelations

→ velmi používán (Google, Yahoo, ...)

1. Agent - gr:BusinessEntity

a. person or organization

2. Object or service - gr:ProductOrService

a. camcorder, house, car

b. haircut

3. Promise (offer) - gr:Offering

a. To transfer some rights (ownership, usage, license) on the object or

b. To provide the service for a certain compensation (money)

c. Made by the agent and related to the object or service

4. Location - gr:Location

a. From which this offer is available

i. store, bus stop, gas station

#### GoodRelations - gr:BusinessEntity

```
foo:ACME a gr:BusinessEntity;
  gr:legalName "ACME Bagel Bakery Ltd."@en;
  foaf:page <http://www.example.com/>;
  s:address [ a s:PostalAddress;
    s:streetAddress "Bagel Street 1234";
    s:postalCode "12345";
    s:addressLocality "Munich, Germany" ];
  s:telephone "+49-89-12345678-0";
  s:faxNumber "+49-89-12345678-99";
  s:email "contact@example.org".
```

```
foo:restaurant a gr:Location;
  gr:name "Hepp's Happy Burger Restaurant"@en;
  gr:hasOpeningHoursSpecification
    [ a gr:OpeningHoursSpecification;
      gr:opens "08:00:00"^^xsd:time;
      gr:closes "12:00:00"^^xsd:time;
      gr:hasOpeningHoursDayOfWeek gr:Monday,
      gr:Tuesday, gr:Wednesday, gr:Thursday,
      gr:Friday ],
    [ a gr:OpeningHoursSpecification;
      gr:opens "13:00:00"^^xsd:time;
      gr:closes "20:00:00"^^xsd:time;
      gr:hasOpeningHoursDayOfWeek
      gr:Friday ] .
```

#### → gr: OpeningHoursSpecification

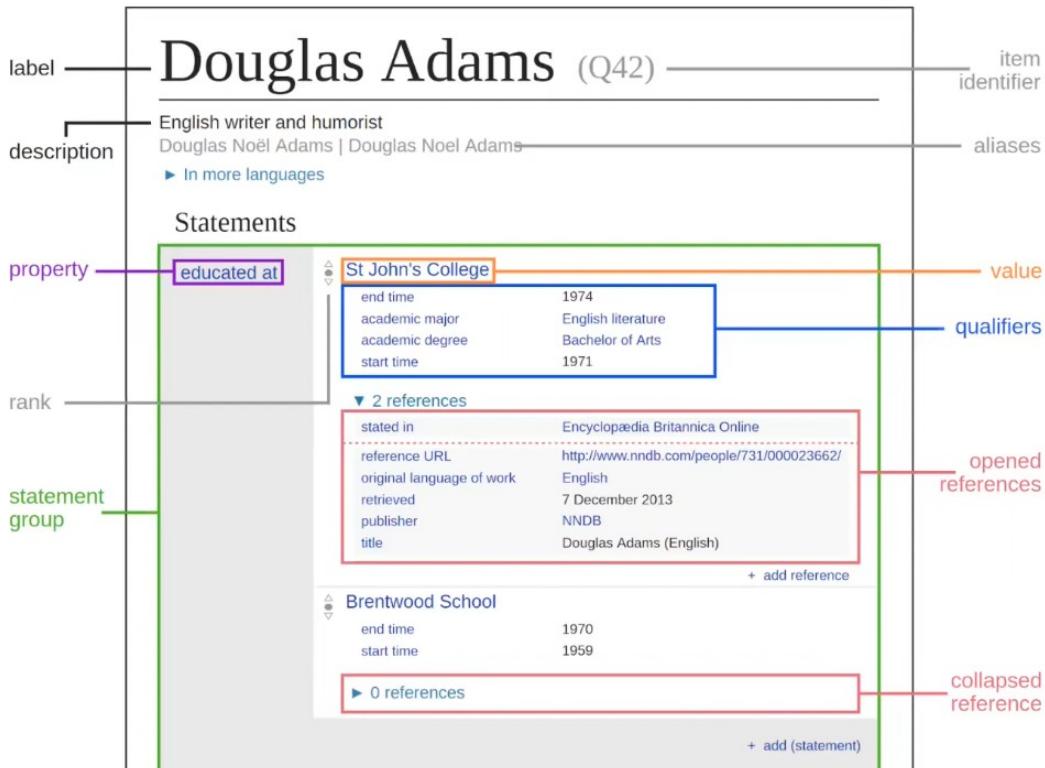
- A real product
  - e.g. my laptop with its serial number
  - e.g. my car with its VIN and mileage
  - e.g. a particular item on eBay
  - Can be sold only once
  - gr:Individual
- A product model
  - Nikon T90, iPod Nano 16 GB
  - Abstract definition, not a particular item
  - In GoodRelations modeled as prototype
  - In contrast to modeling as datasheet
  - gr:ProductOrServiceModel

-- nebo „someItems“

**Schema.org** → spojuje různé existující slovníky a bere si z nich classy

## WikiData

- dotazovatelné přes SPARQL
- rozdělené na „položky“
- dotazuji se do WikiData přes SPARQL endpoint

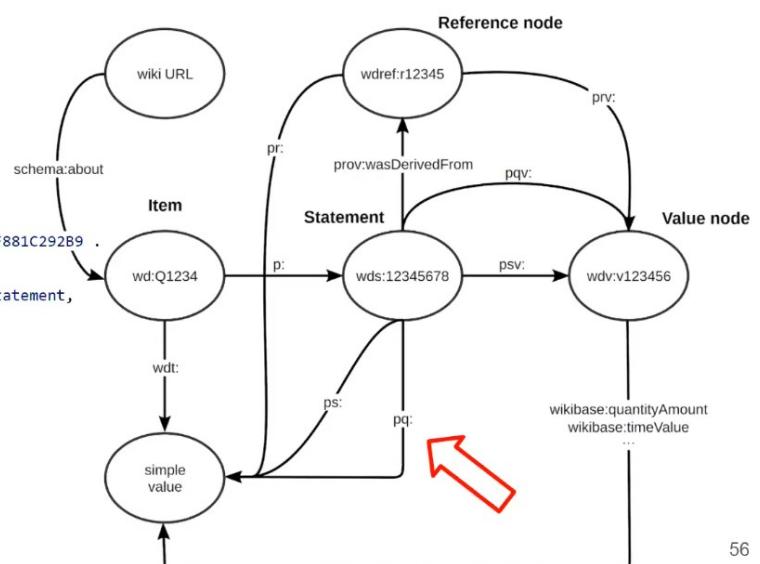


## Wikidata RDF data model - qualifiers

```
@prefix wd: <http://www.wikidata.org/entity/> .
@prefix wikibase: <http://wikiba.se/ontology#> .

wd:Q42 a wikibase:Item .
wd:Q42 rdfs:label "Douglas Adams"@en ;
  #P3373 - sibling
  #Q14623673 - Susan Adams
  wdt:P3373 wd:Q14623673 ;
  wd:Q42 p:P3373 s:Q42-A3B1288B-67A9-4491-A3AA-20F881C292B9 .

s:Q42-A3B1288B-67A9-4491-A3AA-20F881C292B9 a wikibase:Statement,
  wikibase:BestRank ;
  wikibase:rank wikibase:NormalRank ;
  ps:P3373 wd:Q14623673 ;
  #P1039 - kinship to subject
  #Q10943095 - younger sister
  pq:P1039 wd:Q10943095 ;
  prov:wasDerivedFrom
  ref:6a3634133c828f5c3cba3f33d033c4d2ae67f5ec .
```



56

<https://query.wikidata.org/>

The screenshot shows the Wikidata Query Service interface with the following query:

```
1 #Goats
2 SELECT ?item ?itemLabel
3 WHERE
4 {
5   ?item wdt:P31 wd:Q2934.
6   SERVICE wikibase:label { bd:serviceParam wikibase:language "[AUTO_LANGUAGE],en". }
7 }
```

## 5. Přednáška

**LPG** – Labeled Property Graph

→ když chci více řešit vztahy než samotné entity

- může být výhoda mít grafové algoritmy

- **orientovaný multigraf, vrcholy mají sety labelů, hrany mají labely, hrany i vrcholy mají key-value vlastnosti**

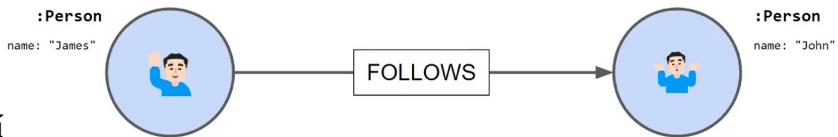
**jako sety**

- nemáme zde schémata

- jiná reprezentace grafu než RDF (a na rozdíl od RDF mohou být vrcholy v LPG seznam hodnot)

- uzly nejsou IRI, jen pojmenovány

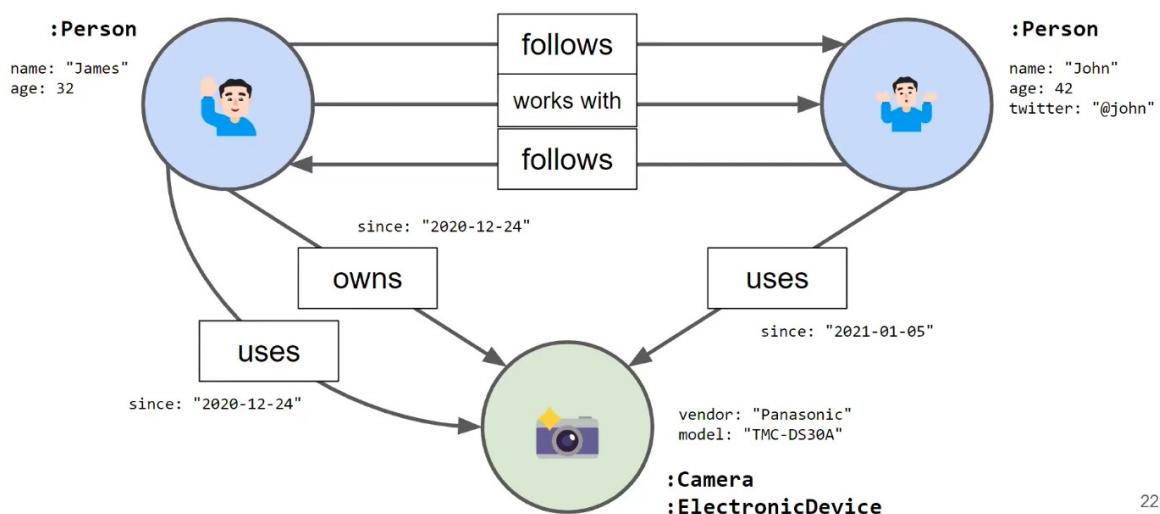
## Simple Cypher create statement



```
CREATE (:Person {name: 'James'})-[FOLLOWES]->(:Person {name: 'John'})
```

## Cypher

## Labeled Property Graph data model



22

## dotazovací jazyk nad LPG

- CRUD operace, neo4j

- fráze:

viz obrázky vpravo

duplicita:

```
//Who uses a camera owned by a followed person?
MATCH
  (p1:Person)-[:USES]->(c:Camera)<-[:OWNS]-(p2:Person)<-[FOLLOWES]-(p1:Person)
RETURN
  p1
```

```
CREATE CONSTRAINT ON (p:Person)
ASSERT p.name IS UNIQUE
```

najde či vytvoří:

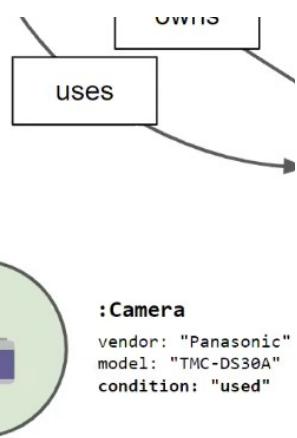
```
MERGE (p:Person {name: 'James'})
CREATE (p)-[:OWNS]->(:Laptop {vendor: 'Dell'})
```

při vytvoření set:

```
MERGE (p:Person {name: 'James'})
ON CREATE SET
  p.twitter = '@james'
MERGE (p)-[:OWNS]->(:Laptop {vendor: 'Dell'})
```

```
//Find James's camera
```

```
MATCH
  (c:Camera)<-[:OWNS]-(p:Person)
WHERE
  p.name = 'James'
SET
  c += {condition: 'used'}
RETURN
  c
```



- Null je missing value
- obsahuje algoritmy  
(shortestPath, ...)

### **- Return \* (vrádí graf) - Return p.Name (tabulka)**

- Agregace --- stačí Return
- klasické operátory fungují

### **- WITH (manipulace s proměnnou)**

## **GDS – algoritmy, knihovna pro neo4j**

### **GQL Standard?**

→ ještě není. (bude asi OpenCypher)

## Cypher - AsciiArt

Nodes  
(`()` (`p`))

Labels start with `:` and represent types  
(`p:Person:Mammal`)

Nodes can have properties  
(`p:Person {name:'John'}`)

Relationships  
`--> -[f:FOLLOWS]-`

Direction of relationship can go both ways  
(`(p1)-[:FOLLOWS]->(p2)` or `(p1)<-[:FOLLOWS]-(p2)`)

Relationships can have properties too  
`-[:OWNS {since: '2021-02-21'}]->`

## Cypher style guide

### Node labels

- Case sensitive
- `UpperCamelCase`  
i.e. beginning with an upper-case character
- `:VehicleOwner` rather than  
`:vehicle_owner` etc.

### Property keys

- Case sensitive
- `lowerCamelCase`
- `twitterHandle` rather than `TwitterHandle`

### Cypher keywords

- Case insensitive
- However, recommended to use upper-case
- `MATCH` rather than `MaTcH`

### Relationship types

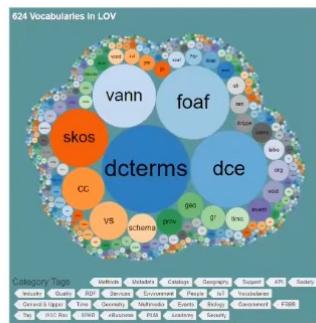
- Case sensitive
- `SCREAMING_SNAKE_CASE`, i.e. upper-case, using underscore to separate words
- `:OWNS_VEHICLE` rather than  
`:ownsVehicle` etc.

## Porovnání RDF a LPG

### Resource Description Framework (RDF)

Made for the Web, distributed data

- global identification: IRIs for everything
- globally reused RDF vocabularies
  - focus on meaning of data
- focused on linking data from various publishers



### Labeled Property Graph (LPG)

Made for centralized graph data

- local node labels and edge types
- every database instance uses different relationship types and node labels
- better at evaluating graph algorithms
  - e.g shortest path

RDF alá LPG jednoduchost:

### RDF-star - RDF\*

Emerging technique, see the [latest editor's draft](#).

Allows RDF statements to appear as subjects and objects of other RDF statements.

Based on RDF reification, but weaker.

The triple in the subject or object position is enclosed in `<>`, called "quoted triple".

### ---- RDF + quoted triple

```
#Bob states: there is someone named Alice
#There is someone named Alice, stated Bob.
<< _:a :name "Alice" >> :statedBy :bob.

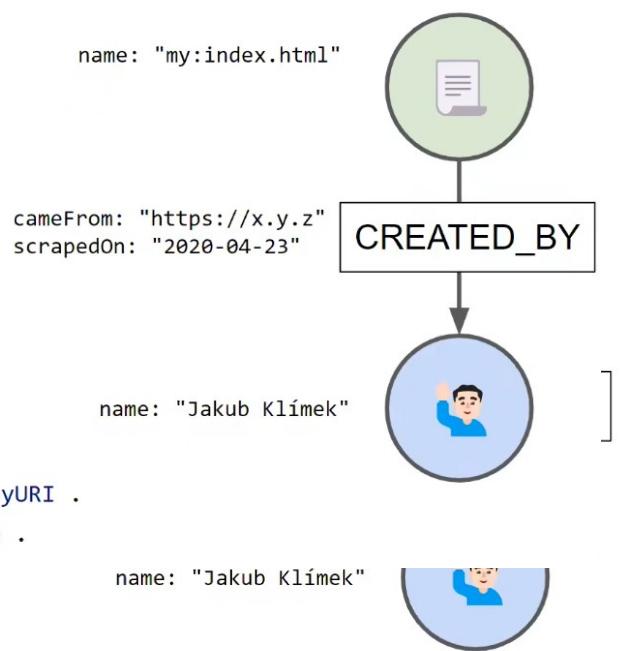
#Employee22 claims that employee38's job title is
Assistant designer
:employee38 :familyName "Smith" .
:employee22 :claims
<< :employee38 :jobTitle "Assistant Designer" >> .
```

### + SPARQL\*

## Resource Description Framework (RDF)

```
my:index.html my:createdBy "Jakub Klímek" .  
  
_:triple1 a rdf:Statement .  
_:triple1 rdf:subject my:index.html .  
_:triple1 rdf:predicate my:createdBy .  
_:triple1 rdf:object "Jakub Klímek" .  
_:triple1 dcterms:source "https://x.y.z"^^xsd:anyURI .  
_:triple1 dcterms:created "2020-04-23"^^xsd:date .  
dcterms:created "2020-04-23"^^xsd:date .
```

## Labeled Property Graph (LPG)



vs. **RDF\***:

## 6. Přednáška HIERARCHICKÉ DATOVÉ MODELY

### XML

- = extensible markup language
- > dokumentově-centrické XML
- > **datově orientované XML**

### - CASE SENSITIVE!

- první řádek XML = prolog  
(kódování a verzi XML, utf-8)
- začíná **kořenem** (pouze jeden root el.)
- elementy mají volitelné **atributy**
- vše má start a end tag

<!—komentář -->

- *mixed content* = text i nested elements
- *well-formed* = pokud splňuje všechno výše

- *namespaces* (např. <h:table>)
  - > definuji v rootu

```
<root xmlns:h="http://www.w3.org/TR/html4/"  
      xmlns:f="https://www.w3schools.com/furniture">  
  <h:table>  
    <h:tr>  
      <h:td>Apples</h:td>  
      <h:td>Bananas</h:td>
```

- CDATA sekce  
(možnost string se special znaky)

```
<?xml version="1.0" encoding="UTF-8"?>  
<elementA>  
  <![CDATA[<greeting>Hello, world!</greeting>]]>  
</elementA>
```



- *lang attribute*  
<p xml:lang="en">

- XML PI (processing instruction)  
<?xmlstylesheet type="text/xsl" href="style.xsl"?>

- **entity** (pro vynechávání special znaků) + lze definovat vlastní entity (DTD)  
< &lt; < &gt;

Principy:

**DOM** (document object model) -> celé XML si načte do paměti, pro HTML ideální na náhodný přístup

**SAX** (simple api for xml) -> velká XML, parsuje element po elementu, ne na random přístup

**STAX** -> streaming SAX, „a zpracuj další element, skonči, jakmile to najdeš“

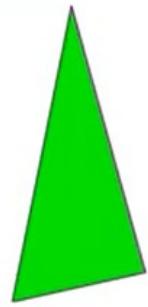
```
Dear <customer>John Doe</customer>,  
  
the balance on your bank account <accountNumber>111333444/1123</accountNumber> as of  
<balanceDate>3rd of January 2021</balanceDate> is <balance>25000 CZK</balance>.  
  
Best regards,  
  
<bankName>Your bank</bankName>  
<streetAddress>1234 5th Avenue</streetAddress>  
<phone>+420123456789</phone>
```

```
<?xml version="1.0" encoding="UTF-8"?>  
<catalog>  
  <title>My catalog</title>  
  <description>This is my dummy catalog</description>  
  <contact-point>  
    <name>John Doe</name>  
    <e-mail>mailto:john@doe.org</e-mail>  
  </contact-point>  
  <datasets>  
    <dataset>  
      <title>My first dataset</title>  
    </dataset>  
    <dataset>  
      <title>My second dataset</title>  
    </dataset>  
  </datasets>  
</catalog>
```

dotazovací jazyky nad XML -> XPath, XQuery  
transformace do jiných formátů – XSLT

```
<svg height="210" width="500">
<polygon points="200,10 250,190 160,210"
style="fill:lime;stroke:purple;stroke-width:1" />
</svg>
```

SVG -> vektorová logika pomocí XML jako hostitelský formát  
OOXML – open office xml  
Atom, RSS – šlo mít takto nastavené posílaní novinek



## XML schéma

- má XML jako hostitelský formát

### XSD and XML documents / instances

```
<?xml version="1.0" encoding="utf-8"?>
<xss:schema xmlns:xss="http://www.w3.org/2001/XMLSchema">
... <!-- XML schema definition --> ...
</xss:schema>
```

```
<?xml version="1.0" encoding="utf-8"?>
<root_element_of_XML_document
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="schema2.xsd">
... <!-- XML document --> ...
</root_element_of_XML_document>
```

Link to XML Schema

-> **XML dokument validní, pokud má schéma a je vůči němu validní**

- je dobré postupovat dle doporučení
- simple types: bool, anyURI, ...
- definuje datové typy, elementy, skupiny

### simple type

-> xs:restriction (aspoň 42)

### complex type --- má atributy

#### XSD - Complex type - simple content + attribute

```
<?xml version="1.0" encoding="UTF-8"?>
<xss:schema xmlns:xss="http://www.w3.org/2001/XMLSchema">
<xss:element name="Catalog">
<xss:complexType>
<xss:simpleContent>
<xss:extension base="xs:string">
<xss:attribute name="numberOfPublishers"
  type="xs:integer"
  use="required"/>
</xss:extension>
</xss:simpleContent>
</xss:complexType>
</xss:element>
</xss:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<Catalog
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="Schema06.xsd"
  >22</Catalog>
```



```
<?xml version="1.0" encoding="UTF-8"?>
<Catalog
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="Schema06.xsd"
  numberOfPublishers="42"
  >someText</Catalog>
```

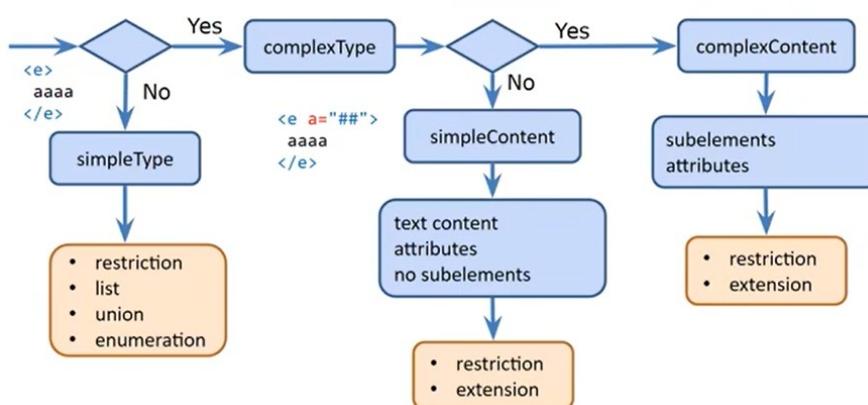


64

subelements  
or attributes?

subelements?

```
<e a="##">
<e2></e2>
<e3></e3>
</e>
```



**choice** (jedna z možností) vs **sequence** (vše, přesné pořadí) vs **all** (musí tam být vše, pořadí je jedno)

**qualified elementy** -> vyhne se kolizi s jiným schématem

## 7. přednáška

### XPath

-> dotazovací jazyk nad XML

--- přístupové funkce: text()

--- filtrování

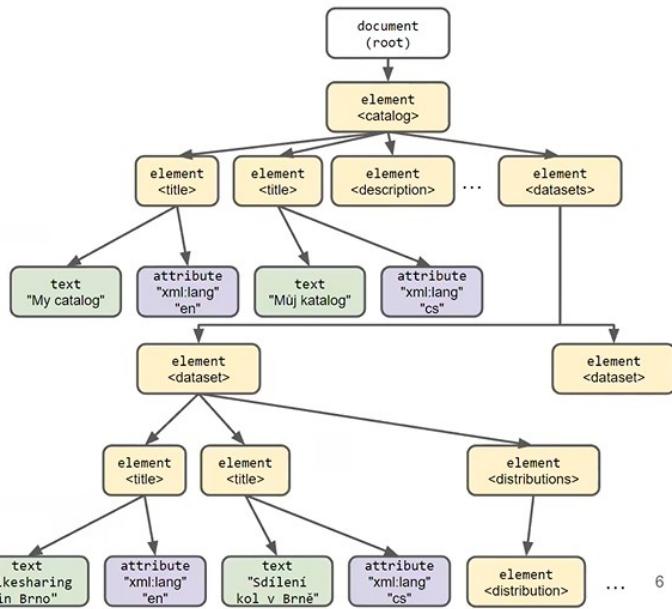
- zavedený model pro XML = XPath datový model

Elementy XML mají obsah

XML schéma typy

### XPath Data Model

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
  <title xml:lang="en">My catalog</title>
  <title xml:lang="cs">Můj katalog</title>
  <description xml:lang="en">This is my dummy catalog</description>
  <description xml:lang="cs">Toto je můj falešný katalog</description>
  <contact-point>
    <name xml:lang="en">John Doe</name>
    <e-mail>john@doe.org</e-mail>
  </contact-point>
  <datasets>
    <dataset>
      <title xml:lang="en">Bikesharing in Brno</title>
      <title xml:lang="cs">Sdílení kol v Brně</title>
      <distributions>
        <distribution>
          <media-type>application/xml</media-type>
          <downloadURL>http://brno.cc/myfile.xml</downloadURL>
        </distribution>
        <distribution>
          <accessService>
            <endpointURL>https://brno.cz/myAPI</endpointURL>
            <title xml:lang="en">My API</title>
            <accessService>
          </distribution>
        </distribution>
      </distributions>
    </dataset>
    <dataset>
      <title xml:lang="en">Bikesharing in Prague</title>
      <title xml:lang="cs">Sdílení kol v Praze</title>
      <distributions>
        <distribution>
          <title xml:lang="en">CSV</title>
          <media-type>text/csv</media-type>
          <downloadURL>http://praha.eu/myfile.csv</downloadURL>
        </distribution>
      </distributions>
    </dataset>
  </datasets>
</catalog>
```

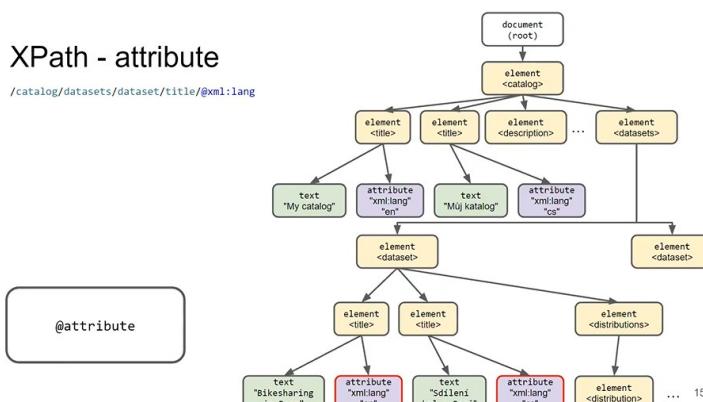


6

příklad (atribut):

### XPath - attribute

/catalog/datasets/dataset/title/@xml:lang



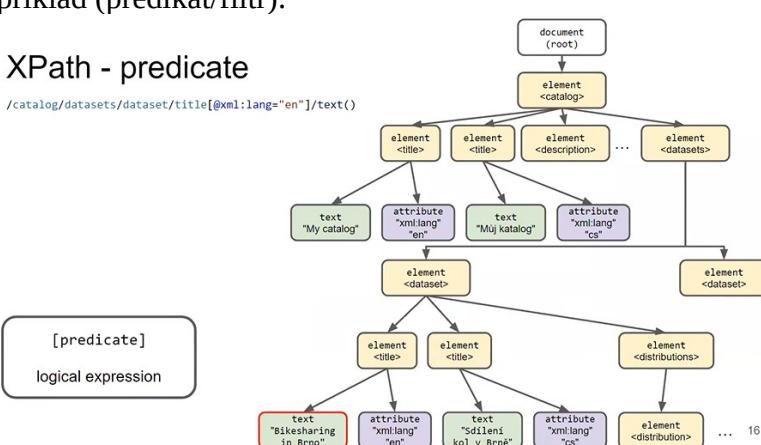
@attribute

15

příklad (predikát/filtr):

### XPath - predicate

/catalog/datasets/dataset/title[@xml:lang="en"]/text()



[predicate]

logical expression

16

- můžeme postupovat **relativní cestou** (dotaz bez lomítka na začátku)

- **axis**: osy dle toho, jakým způsobem chci hledat  
child = defaultní (vrací všechny elementy)

`/catalog/child::*`

`/catalog/*`

descendant = všechny potomci splňující něco

-> může však vypsat i věci „níže“ se stejným názvem

`/catalog/descendant::title`

attribute = vypisuji atributy

`/catalog/descendant::*[attribute::*]`

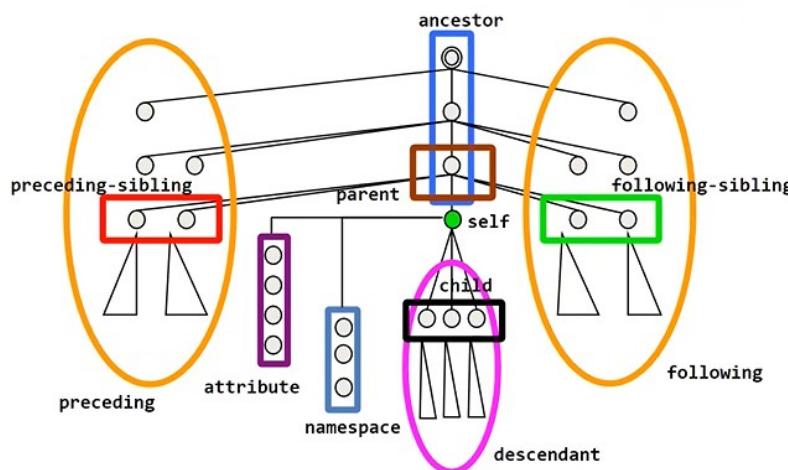
`/catalog/descendant::*[@*]`

preceding sibling -> předchozí sourozenec

descendant-or-self -> dvojité lomítko „//“, obsahuje i to, ze které se ptám

self -> „..“

parent -> obsahuje rodiče elementu, kde zrovna jsem „..“



## Funkce

`position()` -> vrací číslo pozice      `name()` -> vrací název elementu

`last()` -> vrací počet elementů v uzlu

Axes		Node tests	
<code>self</code>		<code>name</code>	node with particular <code>name</code>
<code>child</code>		<code>*</code>	node with arbitrary name
<code>descendant</code>		<code>node()</code>	any node
<code>parent</code>		<code>text()</code>	any text node
<code>ancestor</code>		Functions	
<code>attribute</code>		<code>position()</code>	position of node in the result
<code>following(-siblings)</code>		<code>last()</code>	position of the last node in the result
<code>preceding(-siblings)</code>		<code>count()</code>	number of nodes in the result
<code>normalize-space()</code>	normalization of white spaces		
<code>name()</code>	name of node		
Abbreviations			
<code>.../...</code>	<code>../child::...</code>		
<code>.../@...</code>	<code>../attribute::...</code>		
<code>.../...</code>	<code>./self::node()...</code>		
<code>.../...</code>	<code>./parent::node()...</code>		
<code>.../...</code>	<code>./descendant-or-self::node()...</code>		

## XSLT (XSL transformace)

-> jazyk transformující XML do HTML, RDF (a mnoho jiných formát)

- hostitelský jazyk XML

-> XPath je důležitý pro XSLT

-> **XSLT Stylesheet** – xml dokument obsahující šablony (templates)

**template** -> matchuje část input XML pomocí XPATH

**processor** -> prochází XML dokument, matchuje templaty

= podle stylesheetu vyprodukuje výstup

## XSLT - empty stylesheet

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:fn="http://www.w3.org/2005/xpath-functions">

  <xsl:output method="html" encoding="UTF-8" indent="yes" />

</xsl:stylesheet>
```

match -> co musí být splněno (skrze XPath výraz)

xsl:template -> jde na výstup

xsl:value-of -> hodnotu do výstupu (např. název daného catalogu)

## XSLT - second template

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:fn="http://www.w3.org/2005/xpath-functions">

  <xsl:output method="html" encoding="UTF-8" indent="yes" />
  <xsl:template match="catalog">...</xsl:template>

  <xsl:template match="dataset">
    <h2>
      <xsl:value-of select="title[@xml:lang='en']"/>
    </h2>
    <p>
      Number of distributions:
      <xsl:value-of select="count(descendant::distribution)"/>
    </p>
  </xsl:template>

</xsl:stylesheet>
```

```
<html
  xmlns:fn="http://www.w3.org/2005/xpath-functions"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>My catalog</title>
  </head>
  <body>
    <h1>My catalog</h1>
  </body>
</html>
```

Nothing new in the output... why?

-> zde nutné přidat xsl:apply-templates !

-- implicitní šablony (nějaké předdefinované šablony matchující texty, elementy a další...)

- pojmenovaní šablon:

## XSLT - named templates and parameters

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:fn="http://www.w3.org/2005/xpath-functions"
  xmlns:util="http://www.w3.org/2002/util/transforms"
  <xsl:output method="html" encoding="UTF-8" indent="yes" />
  <xsl:strip-space elements="*"/>
<xsl:template match="catalog">
  <xsl:call-template name="processTitle">
    <xsl:with-param name="element" value="title"/>
    <xsl:with-param name="lang" value="en"/>
  </xsl:call-template>
</xsl:template>
<xsl:template match="dataset">
  <xsl:call-template name="processTitle">
    <xsl:with-param name="element" value="title"/>
    <xsl:with-param name="lang" value="en"/>
  </xsl:call-template>
  <xsl:apply-templates select="datasets/dataset" />
</xsl:template>
<xsl:template match="text()"/>
<xsl:template name="processTitle">
  <xsl:param name="element" required="yes"/>
  <xsl:param name="lang" required="yes"/>
  <xsl:element name="{$element}">
    <xsl:value-of select="title[@xml:lang=$lang]"/>
  </xsl:element>
</xsl:template>
<xsl:element name="dataset">
  <xsl:apply-templates select="dataset" />
</xsl:element>
```

### Named templates

- name attribute instead of match attribute
- accept parameters
  - **xsl:param** - definition in named template
  - **\$variable** - access to variable value in XPath
  - **{\$variable}** - access to variable value elsewhere
- called using **xsl:call-template**
  - does not change the currently processed node set
  - **xsl:with-param** - values passed when calling

### xsl:element

- creates an element on the output
- name can be constant or **{\$variable}**

- lze nastavit i globální proměnné, podmínky a další (včetně switch a for each)

## XSLT - global variables, modes, if

```
<xsl:stylesheet version="1.0" encoding="UTF-8">
<xsl:stylesheet version="1.0"
    xmlns="http://www.w3.org/1999/XSL/Transform"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:fn="http://www.w3.org/2005/xpath-functions"
    xmlns:output="http://www.w3.org/1999/XSL/Format"
    output-method="html" encoding="UTF-8" indent="yes" />
<xsl:variable name="lang">en</xsl:variable>

<xsl:template match="catalog">
    <xsl:choose>
        <xsl:when>
            <xsl:apply-templates mode="head"/>
        <xsl:otherwise>
            <xsl:apply-templates mode="catalog"/>
        </xsl:otherwise>
    </xsl:choose>
</xsl:template>

<xsl:template match="dataset" mode="head"/>

<xsl:template match="dataset" mode="catalog">
    <xsl:apply-templates mode="dataset"/>
    <xsl:value-of select="count(descendant::distribution)"/>
</xsl:template>

<xsl:template match="title" mode="head">
    <xsl:if test="@xml:lang=$lang">
        <xsl:element name="title">
            <xsl:value-of select="text()"/>
        </xsl:element>
    </xsl:if>
</xsl:template>

<xsl:template match="title" mode="catalog">
    <xsl:if test="@xml:lang=$lang">
        <xsl:element name="hi">
            <xsl:value-of select="text()"/>
        </xsl:element>
    </xsl:if>
</xsl:template>

<xsl:template match="text()" mode="#all"/>
</xsl:stylesheet>
```

### Global variable

- defined in the `xsl:stylesheet` root element using `xsl:variable`
- accessible in the whole stylesheet
  - e.g. `$lang`

### Mode

- ability to process the same nodes in different ways
  - different templates with the same `match`
- specified in `xsl:apply-templates`
- used in unnamed `xsl:template`
  - `#all` matches all modes

## 8. Přednáška (stále hierarchické modely)

**JSON** – javaScript object notation

-> především UTF-8, escapes „ \ “

- bílé znaky = jakékoliv (konvence)

- JSON čísla: desítková, desetinná hodnota pomocí tečky (3.14)

-> **hodnoty**: string, number, object („{}“), array (hranaté závorky), bool, null

**JSON Lines** – validní JSON po řádcích (něco mezi CSV a JSON) --- proč? zanořování.

- lze konvertovat JSON do XML pomocí XSLT

**JSON pointer** – funguje jako cesta, která ukazuje dovnitř do souboru

**JSON schéma**

- slovník pro JSON

- určuje jistá pravidla, lepší čitelnost

--- klíčová slova označená \$, přesné hodnoty přes properties

## JSON Schema

<https://json-schema.org/>

Proposed IETF standard

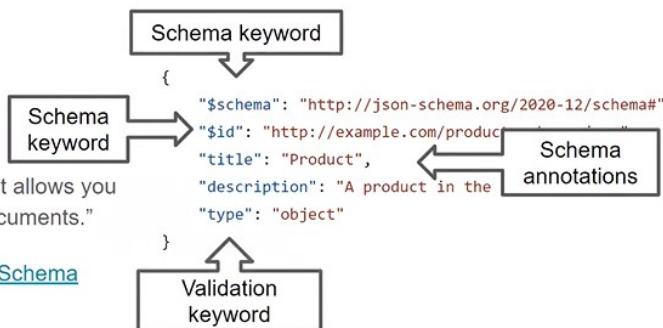
"**JSON Schema** is a vocabulary that allows you to **annotate** and **validate** JSON documents."

Online book: [Understanding JSON Schema](#)

First draft: 2009

Still under development

```
{  
    "productId": 1,  
    "productName": "A green door",  
    "price": 12.5,  
    "tags": [ "home", "green" ]  
}
```



✓ No errors found. JSON validates against the schema.

## JSON Schema - properties

```
{  
    "productId": 1,  
    "productName": "A green door",  
    "price": 12.5,  
    "tags": [ "home", "green" ]  
}  
  
{  
    "$schema": "http://json-schema.org/2020-12/schema#",  
    "$id": "http://example.com/product.schema.json",  
    "title": "Product",  
    "description": "A product from Acme's catalog",  
    "type": "object",  
    "properties": {  
        "productId": {  
            "description": "The unique identifier for a product",  
            "type": "integer"  
        },  
        "productName": {  
            "description": "Name of the product",  
            "type": "string"  
        }  
    },  
    "required": [ "productId", "productName" ]  
}
```

- plus máme constrains jako: multipleOf, maximum, ...

- validace listů (bud' jako listy nebo tuple) --- plus můžu vypnout additionalItems (všechny definované věci nemusíme obsáhnout)

## JSON Schema - JSON arrays - tuple validation

Each item in the array has different schema

Positions are meaningful

- ✓ [1600, "Pennsylvania", "Avenue", "NW"]
- ✗ [24, "Sussex", "Drive"]
- ✗ ["Palais de l'Élysée"]
- ✓ [10, "Downing", "Street"]
- ✓ [1600, "Pennsylvania", "Avenue", "NW", "Washington"]

```
{
  "type": "array",
  "items": [
    { "type": "number" },
    { "type": "string" },
    {
      "type": "string",
      "enum": ["Street", "Avenue", "Boulevard"]
    },
    {
      "type": "string",
      "enum": ["NW", "NE", "SW", "SE"]
    }
  ]
}
```

- zanořené JSON objekty

```
{
  ...
  "properties": {
    ...
    "dimensions": {
      "type": "object",
      "properties": {
        "length": { "type": "number" },
        "width": { "type": "number" },
        "height": { "type": "number" }
      },
      "required": [ "length", "width", "height" ]
    }
  }
  ...
}
```

- použití externího JSON schématu

```
{
  ...
  "properties": {
    ...
    "warehouseLocation": {
      ...
      "description": "Coordinates of the warehouse ...",
      "$ref": "https://example.com/geographical-location.schema.json"
    }
  }
}
```

- má string contrains

+ základní JSON datové formáty

<pre>{   "type": "string",   "minLength": 2,   "maxLength": 3 }</pre>	<p>built-in formats for strings</p> <ul style="list-style-type: none"> <li>• date-time, date, time, duration</li> <li>• email, idn-email</li> <li>• hostname, idn-hostname</li> <li>• ipv4, ipv6</li> </ul> <p>"typ_turistikého_cíle": {     "type": "string",     "format": "iri",     "pattern": "^https://data\\.mvcr\\.gov\\.cz/zdroj/čiselníky/typy-turistických-cílů/položky/.*\$",     "title": "Typ turistikého cíle",     "examples": [       "https://data.mvcr.gov.cz/zdroj/čiselníky/typy-turistických-cílů/položky/přírodní"     ]   }</p> <ul style="list-style-type: none"> <li>• uri, uri-reference</li> <li>• iri, iri-reference</li> <li>• uuid</li> <li>• uri-template</li> <li>• json-pointer, relative-json-pointer</li> <li>• regex</li> </ul>
---	--

- validace vůči kombinaci schémat:

Schemas can be combined

- **allOf**

- valid against all schemas
- can be used for extending schemas

```
"allOf": [
  { "$ref": "http://example.com/address.json" },
  { "properties": {
      "type": { "enum": [ "residential", "business" ] }
    }
  }
]
```

```
{
  "street_address": "1600 Pennsylvania Avenue NW",
  "city": "Washington",
  ✓ "state": "DC",
  "type": "business"
}
```

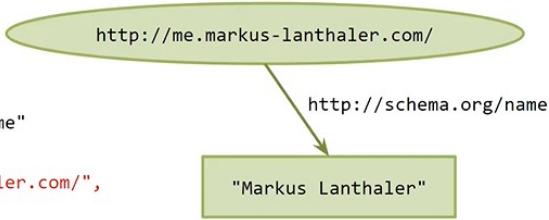
### + validátory na JSON schémata

#### JSON-LD

- JSON pro propojená data (linked)

- @ = klíčová slova

```
{
  "@context": {
    ...
    "name": "http://schema.org/name"
  },
  "@id": "http://me.markus-lanthaler.com/",
  "name": "Markus Lanthaler",
  ...
}
```



- jakého je typu? @type

```
{
  "@context": {
    ...
    "Restaurant": "http://schema.org/Restaurant",
    "Brewery": "http://schema.org/Brewery"
  },
  "@id": "http://example.org/places#BrewEats",
  "@type": [ "Restaurant", "Brewery" ],
  ...
}
```

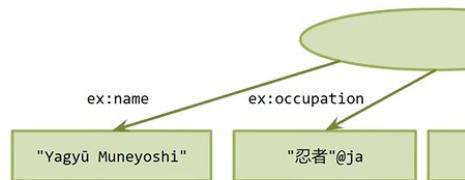
- v kontextu si definuju **prefixy**

- **embedding** = používání objekt jako hodnotu jiného objektu

- scoped context = definuje kontext uvnitř některých objektů

- jazyky (např. přes **language map**):

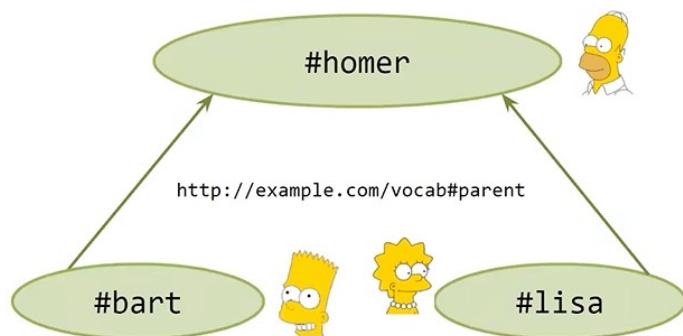
```
{
  "@context": {
    ...
    "occupation": { "@id": "ex:occupation", "@container": "@language" },
    "name": "Yagyū Muneyoshi",
    "occupation": {
      "ja": "忍者",
      "en": "Ninja",
      "cs": "Nindža"
    }
    ...
  }
}
```



@List -> zachováme pořadí jako v RDF

## JSON-LD - reverse properties 2/2

```
{
  "@context": {
    "name": "http://example.com/vocab#name",
    "children": { "@reverse": "http://example.com/vocab#parent" }
  },
  "@id": "#homer",
  "name": "Homer",
  "children": [
    {
      "@id": "#bart",
      "name": "Bart"
    },
    {
      "@id": "#lisa",
      "name": "Lisa"
    }
  ]
}
```



## 9. Přednáška RELAČNÍ DATOVÉ MODELY

### Relační datový model + relační schéma

**SQL Dump** → script v SQL, vytvoří celou databázi

DSV → delimiter seperated values (předchůdce CSV)

TSV → tab seperated values, řádky oddělené new lines

### CSV

- *comma seperated values*
- CSV je RFC 4180 (specifikace)
- **CRLF jako konec řádku** vždy
- **escapování:** uvozovky („)
- hlavička dobrovolná, Null = null
- věci jako hodnota a jednotka (5t) rozdělit na dva sloupce (UN/CEFACT)
- 
- v Excelu je problém, že to ukládá se středníky a UTF-8 BOM (radší Google Sheets)

Data types based on XML Schema (XSD), e.g.

- xsd:boolean
  - ✓ true, false
  - ✗ ANO, yes, 1
- xsd:integer
  - ✓ 1, 2, 3, 4, -1, -2, 0, 2000000
  - ✗ 1-, 2-, "2 000 000"
- xsd:decimal
  - ✓ 1.1, 1.8, -94.4
  - ✗ 1,4, 20,4

- xsd:date
  - ✓ 2021-02-15
  - ✗ 9.4.2017
- xsd:time
  - ✓ 09:00:00, i.e. HH:MM:SS
  - ✗ 04:20, 4:56, 24
- xsd:dateTime
  - ✓ YYYY-MM-DDTHH:MM:SS
  - i.e. xsd:date + "T" + xsd:time

- datové typy:

dva přístupy:

### CSV Styles - one big CSV vs. multiple CSVs

One big CSV - redundant, some prefer it

```
Airport,Continent,Month,Condition,Value
PRG,Europe,January,Rain,0
LHR,Europe,January,Rain,20
DXB,Asia,January,Rain,0
PRG,Europe,January,Snow,20
LHR,Europe,January,Snow,0
DXB,Asia,January,Snow,0
PRG,Europe,February,Rain,0
LHR,Europe,February,Rain,20
DXB,Asia,February,Rain,0
PRG,Europe,February,Snow,30
LHR,Europe,February,Snow,0
DXB,Asia,February,Snow,0
PRG,Europe,March,Rain,0
```

Airport,Month,Condition,Value

```
PRG,January,Rain,0
LHR,January,Rain,20
DXB,January,Rain,0
PRG,January,Snow,20
LHR,January,Snow,0
DXB,January,Snow,0
PRG,February,Rain,0
LHR,February,Rain,20
DXB,February,Rain,0
PRG,February,Snow,30
LHR,February,Snow,0
DXB,February,Snow,0
PRG,March,Rain,0
```

Airport,Continent

```
PRG,Europe
LHR,Europe
DXB,Asia
```

### CSV on Web:

- anotace CSV tabulek pomocí json-ld
- URI fragmenty identifikují CSV soubor (lze se zeptat na daný řádek)
- skupina tabulek (více CSV souborů) nějak propojené
- jde vkládat i části xml, **base, format, minimum, length**
- je dobré specifikovat „name“

Table schema describes columns, rows and cells

```
{
  "@context": "http://www.w3.org/ns/csvw",
  "@type": "Table",
  "@id": "https://example.org/table1",
  "url": "https://example.org/table1.csv",
  "tableSchema": {
    "columns": [
      {
        "titles": "airport",
        "dc:description": "An identifier for the airport.",
        "datatype": "string",
        "required": true
      },
      {
        "titles": "continent",
        "dc:description": "The continent the airport is on.",
        "datatype": "string",
        "lang": "en",
        "required": true
      }
    ]
  }
}
```

airport	continent
PRG	Europe
DXB	Asia

propojení s dalšími CSV:

## CSVW - JSON-LD descriptor - Table schema

**foreignKeys** between tables is an array of key references to tables in the same table group

```
{
  "@context": "http://www.w3.org/ns/csvw",
  "tables": [
    {
      "url": "https://example.org/table1.csv",
      "tableSchema": {
        "columns": [
          {
            "name": "airport",
            "titles": "letiště",
            "datatype": "string"
          },
          {
            "name": "continent",
            "titles": "kontinent",
            "datatype": "string"
          }
        ],
        "primaryKey": "airport"
      }
    }
  ]
}
```

letiště	kontinent
PRG	Europe
DXB	Asia

```
{
  "url": "https://example.org/table2.csv",
  "tableSchema": {
    "columns": [
      {
        "name": "airfield",
        "titles": "airfield",
        "datatype": "string"
      },
      {
        "titles": "month",
        "datatype": "string"
      },
      {
        "titles": "snow",
        "datatype": "number"
      }
    ],
    "primaryKey": "airfield",
    "foreignKeys": [
      {
        "columnReference": "airport",
        "reference": {
          "resource": "https://example.org/table1.csv",
          "columnReference": "airport"
        }
      }
    ]
  }
}
```

airfield	month	snow
PRG	January	20
DXB	January	0

53

(lze i na úrovni schématu)

primaryKey = využíj „name“, ne „title“

- CSV dialect

- možnost předefinovat kódování, nastavení, konce řádků
- možnost seznamů v hodnotách, vlastní čas, vlastní NULL hodnota

- JSON-LD deskriptor pro CSV soubor = {název\_csv}-metadata.json

- generování RDF z tabulkárních dat

- 1. Simple String Expansion: {var}
  - Variable expansion of a defined, non-empty value results in a substring of allowed URI characters.
  - Result is URI, not IRI. Percent-decode by implementation is necessary to obtain an IRI.
- 2. Reserved Expansion: {+var}
  - same as Simple String Expansion
  - in addition, variable can contain percent-encoded triplets and reserved URI characters
- 3. Fragment Expansion: #{var}
  - same as Reserved Expansion
  - in addition, the result is prefixed with '#'

## CSVW - Generating RDF - default conversion

<http://example.org/countries.csv>

```
countryCode,latitude,longitude,name
AD,42.5,1.6,Andorra
AE,23.4,53.8,"United Arab Emirates"
AF,33.9,67.7,Afghanistan
```

@base <http://example.org/countries.csv> .

```
_:8228a149-8e7e-448d-b15f-8abf92e7bd1
<#countryCode> "AD" ;
<#latitude> "42.5" ;
<#longitude> "1.6" ;
<#name> "Andorra" .
```

```
_:ec59defc-872a-4144-822b-9ad5e2c6149c
<#countryCode> "AE" ;
<#latitude> "23.4" ;
<#longitude> "53.8" ;
<#name> "United Arab Emirates" .
```

```
_:ef2e8e9-3d02-4bf5-b4f1-4794ba5b52c9
<#countryCode> "AF" ;
<#latitude> "33.9" ;
<#longitude> "67.7" ;
<#name> "Afghanistan" .
```

# CSVW - Generating RDF - customizing conversion

položka, položka\_název\_cs, položka\_název\_en  
<https://data.mff.cuni.cz/zdroj/číselníky/dny-v-týdnu/položky/pondělí>, Pondělí, Monday

Columns annotated by additional properties

- **aboutUrl**
  - URI template for RDF statement subject
- **propertyUrl**
  - URI template for RDF statement predicate
- **valueUrl**
  - URI template for RDF statement object

@prefix skos: <<http://www.w3.org/2004/02/skos/core#>> .  
<<https://data.mff.cuni.cz/zdroj/číselníky/dny-v-týdnu/položky/pondělí>> a skos:Concept .

```
{  
  "@id":  
    "https://data.mff.cuni.cz/soubory/číselníky/dny-v-týdnu.csv",  
  "-metadata.json",  
  "@context": ["http://www.w3.org/ns/csvw", {  
    "@language": "cs" } ],  
  "@type": "Table",  
  "url": "dny-v-týdnu.csv",  
  "tableSchema": {  
    "@type": "Schema",  
    "columns": [{  
      "@type": "Column",  
      "name": "položka",  
      "titles": "položka",  
      "dc:description": "IRI položky",  
      "aboutUrl": "{+polozka}",  
      "propertyUrl": "rdf:type",  
      "valueUrl": "skos:Concept",  
      "required": true,  
      "datatype": "anyURI"  
    }],  
  },  
  "transformations": [  
    {  
      "@type": "Template",  
      "url": "templates/ical.txt",  
      "titles": "iCalendar",  
      "targetFormat":  
        "http://www.iana.org/assignments/media-types/text/calendar",  
      "scriptFormat": "https://mustache.github.io/",  
      "source": "json"  
    }]  
}
```

69

- **virtual** sloupce = označení sloupce neexistujícího v daném CSV souboru

- transformace:

## transformations

- on Table groups and tables
- define how tabular data can be transformed into another format using a script or template
- **url**
  - URL of the script or template
- **scriptFormat**
  - If defined, IANA media type, if not, any URL
- **targetFormat**
  - If defined, IANA media type, if not, any URL
- **source**
  - specifies standard transformation prior to transformation using script or template
  - json, rdf, null
- **titles**

```
{  
  "@context": "http://www.w3.org/ns/csvw",  
  "@type": "Table",  
  "@id": "https://example.org/table10",  
  "url": "https://example.org/table10.csv",  
  "transformations": [  
    {  
      "@type": "Template",  
      "url": "templates/ical.txt",  
      "titles": "iCalendar",  
      "targetFormat":  
        "http://www.iana.org/assignments/media-types/text/calendar",  
      "scriptFormat": "https://mustache.github.io/",  
      "source": "json"  
    }]  
}
```

--- standardně existují validátory, ale lepší jsou offline tools (RDF::Tabular)

## 10. Přednáška

### GEODATA

→ spatial data = jak daleko, jakou cestu, ... ?

--- feature → geometrie

→ další atributy

- **implicitní geodata** = měřitelné věci (poloha, vzdálenost)

- **explicitní geodata** = geografické jména, adresy

- **vektor** → většina, více precizní, rychlejší práce na pc

- **rastr** → papírová mapa, pixely

- **geometrické objekty**: body (restaurace), multi-body (všechny stanice linky metra), čáry (dálnice D1), multi-čáry, polygony (Praha – s hranicemi), povrchy

- **reprezentace geometrie v datech:**

1. WKT = Well-Known Text

→ např.: POINT(50.056 14.434)

2. GML = Geography markup language

```
<gml:Point srsName="http://opengis.net/def/crs/EPSG/0/4326" srsDimension="2">/
  <gml:pos>50.056 14.434</gml:pos>
</gml:Point>
```

```
<gml:Curve srsName="http://opengis.net/def/crs/EPSG/0/5514" srsDimension="2">
  <gml:segments>
    <gml:LineStringSegment>
      <gml:posList>-641126.76 -1093821.18 -641119.35 -1093831.05
                  -641109.75 -1093844.44</gml:posList>
    </gml:LineStringSegment>
  </gml:segments>
</gml:Curve>
```

3. format based

Data format	Geometry representation
GML	GML
GeoJSON	geojson
Shapefile	binary
GeoPackage	SQLite
CSV	any
GeoSPARQL	GML/WKT

a) **GML** → založeno na XML, XSD soubory

b) **GeoJSON** → založeno na JSON, jen wgs-84, vlastní reprezentace geometrie (geojson.io)

c) **ShapeFile** → nativní formát GIS ČR, vícero složek, aby fungovalo

#### Shapefile



#### GeoJSON

JMENO	DATASOU_K	DATASOU_P
1 Jachymka	6260000	vstup do je...
2 Netopyrka	6260000	vstup do je...
3 NULL	6260000	vstup do je...
4 Zbraslavské...	6260000	vstup do je...
5 Černotinsk...	6260000	vstup do je...
6 Výpustek	6260000	vstup do je...
7 Pekárná	6260000	vstup do je...
8 Švédský stál	6260000	vstup do je...
9 Ochozská...	6260000	vstup do je...
10 Cikánkys...	6260000	vstup do je...
11 U Jezevce	6260000	vstup do je...
12 U řída	6260000	vstup do je...
13 Šámalíkovy...	6260000	vstup do je...
14 Špíka	6260000	vstup do je...
15 Ledové slůjky	6260000	vstup do je...
16 Na Turolu	6260000	vstup do je...

```
{
  "geometry": {
    "coordinates": [
      14.419134,
      50.090122
    ],
    "type": "Point"
  },
  "crs": {
    "type": "name",
    "properties": {
      "name": "urn:ogc:def:crs:EPSG::4326"
    }
  },
  "properties": {
    "cislo_orientacni": "22",
    "cislo_popisne": "128",
    "druh_mista": "RESTAURAČNÍ ZAHRÁDKY",
    "druh_zbozi": "",
    "moma": "Praha 1",
    "ulice": "Pařížská"
  },
  "type": "Feature"
}
```

d) **OGC GeoPackage** → SQLite databázový soubor, základní i komplexní geometrické objekty, nejjednodušší a nejfektivnější cesta k uložení geodat (vektor i rastr)

e) **CSV** → velmi jednoduché, nemá doporučovanou geometrii, potřeba escapovat znaky

### - reprezentace geometrie v propojených datech:

1. **Geo WGS-84** → lze reprezentovat data pomocí RDF, funguje však dobře jen pro **body**
2. **GeoSPARQL** → dotazovací jazyk pro **spatial operace**, všechny geometrické objekty (může být moc na malé projekty)
3. **GeoJSON-LD** → jako klasický JSON-LD s přidaným kontextem pro **geojson data**

### Spatial vztahy

- **vztah mezi dvěma** či více **objekty založených na poloze** či **tvaru**
- 1) **topologické** = reprezentován jako funkce vracející **bool** (within, touches, crosses)
- 2) **směrové** = **vlevo/vpravo** od nějaké stacionárního objektu
- 3) **vzdálenostní** = **vzdálenost několika objektu** od sebe
- 4) **temporální** = **vzdáleností vztah** se zakomponovaným **časem**

### Spatial operace = analýza dat nějakým způsobem

- buffer (obtáhnutí, aby všechny body pologynu byly nějak daleko), union, dif, intersect
- clip (odstraní danou věc z původního objektu), distance, convex hull (přetransformuje)

### GIS Software, spatial knihovny

- **QGIS**, PostGIS (pro PostgreSQL), ArcGis, Leaflet (JS mapy), OpenLayers (JS API pro mapy), Mapserver, Geoserver

## 11. Přednáška

### Key-Value Formáty

→ hojně používané pro konfigurační soubory

#### .properties soubor

→ hashtable, jen pro java (existují knihovny)  
+ java specifický encoding

#### Ini soubor

→ inicializační soubor  
→ originálně pro Windows (desktop.ini)  
→ encoding jako .properties, bez specifikace  
→ sekce, rozdělené na **key-value hodnoty**

#### TOML

→ podobné INI, **unicode encoding**  
→ normální typy ve values (string, int, ...)  
- klíče: **bare-keys** (klíč = „hodnota“), quoted („ten klíč“ „hodnota“), dotted (dotted.key = „hodnota“)  
- escaping: \\, víceřádkové: “““  
- čísla: int, hex, nekonečno  
- typy: date, time, ...  
→ podporuje i listy (arrays) a hashtables  
+ arrays of tables:

```
points = [ { x = 1, y = 2, z = 3 },
           { x = 7, y = 8, z = 9 },
           { x = 2, y = 4, z = 8 } ]

# same as:
[[points]]
x = 1
y = 2
z = 3

[[points]]
x = 7
y = 8
z = 9

[[points]]
x = 2
y = 4
z = 8
```

→ clouдовé služby

#### YAML

→ jednoduchý na práci v knihovnami v jazycích  
→ nemá ještě oficiální specifikaci

YAML - sequence / array / list



[ 'Apple', 'Orange', 'Strawberry', 'Mango' ]

→ key-value páry mapping

#### Example: desktop.ini

```
[.ShellClassInfo]
LocalizedResourceName=@%SystemRoot%\system32\shell32.dll,-21770
IconResource=%SystemRoot%\system32\imageres.dll,-112
IconFile=%SystemRoot%\system32\shell32.dll
IconIndex=-235
```

#### Example: dbsettings.ini

```
; last modified 1 April 2001 by John Doe
[owner]
name=John Doe
organization=Acme Widgets Inc.

[database]
; use IP address in case network name resolution is not working
server=192.0.2.62
port=143
file="payroll.dat"
```

# This is a TOML document

```
title = "TOML Example" = 

[owner]
name = "Tom Preston-Werner"
dob = 1979-05-27T07:32:00-08:00

[database]
enabled = true
ports = [ 8000, 8001, 8002 ]
data = [ ["delta", "phi"], [3.14] ]
temp_targets = { cpu = 79.5, case = 72.0 }

[servers]

[servers.alpha]
ip = "10.0.0.1"
role = "frontend"

[servers.beta]
ip = "10.0.0.2"
role = "backend"
```

## TOML - (hash) tables

[table] #table header

```
[table-1]
key1 = "some string"
key2 = 123
```

```
[table-2]
key1 = "another string"
key2 = 456
```

# An employee record

name: Martin D'vloper  
job: Developer  
skill: Elite

mapping:  
key, colon, space, value

- **indentace** !!! odsazení bílým znakem
- multi-řádkový string (viz obrázek vpravo)
- přes knihovny

## YAML - complex mapping keys

? indicates complex mapping key

- sequence
  - mapping

```
-->
?
- "Detroit Tigers"
- "Chicago cubs"
:
- 2001-07-23
?
- "New York Yankees"
- "Atlanta Braves"
:
- 2001-07-02
- 2001-08-12
- 2001-08-14
```

The diagram illustrates the mapping of database rows to objects. Red arrows point from the question mark and colon symbols in the list to the 'Key' and 'Value' labels respectively. The first two items map to 'Key', the third item maps to 'Value', and the remaining four items map to 'Key'.

# Employee records

```
- martin:  
    name: Martin D'veloper  
    job: Developer  
    skills:  
        - python  
        - perl  
        - pascal  
  
- tabitha:  
    name: Tabitha Bitumen  
    job: Developer  
    skills:  
        - lisp  
        - fortran  
        - erlang
```

indentation  
mandatory and  
meaningful!

→ datové typy, tagy, schémata (indikované tagy)  
schéma – má set tagů a jak řešit tagy

```
include_newlines: |  
    exactly as you see  
    will appear these three  
    lines of poetry
```

```
fold_newlines: >  
    this is really a  
    single line of text  
    despite appearances
```

## YAML - core schema

## Core Schema

- extension of JSON schema, towards human readability

```
Regular expression
null | Null | NULL |
/* Empty */
true | True | TRUE |
[-+]? [0-9]+
0o [0-7]+
0x [0-9a-fA-F]+
[-+]? ( \. [0-9]+ | [
[-+]? ( \.inf | \.Inf
\.nan | \.NaN | \.NAN
*
```

```
Resolved to tag
tag:yaml.org,2002:null
tag:yaml.org,2002:null
tag:yaml.org,2002:bool
tag:yaml.org,2002:int (Base 10)
tag:yaml.org,2002:int (Base 8)
tag:yaml.org,2002:int (Base 16)
tag:yaml.org,2002:float (Number)
tag:yaml.org,2002:float (Infinity)
tag:yaml.org,2002:float (Not a number)
tag:yaml.org,2002:str (Default)
```

Použití: Docker compose, GitHub Pages

## YAMI - anchors and aliases

The diagram illustrates the relationship between an anchor and its aliases. At the top, a box labeled "Anchor" has a yellow arrow pointing down to a horizontal line. From this line, two red arrows point left to two boxes labeled "Alias". The first Alias box contains the code: "center: &ORIGIN {x: 73, y: 129}; radius: 7". The second Alias box contains the code: "start: \*ORIGIN finish: { x: 89, y: 102 }". Below these, another horizontal line has two red arrows pointing left to two more boxes labeled "Alias". The first of these contains: "start: \*ORIGIN color: 0xFFEEBB text: Pretty vector drawing.". The second contains: "color: 16772795 start: x: 73 y: 129 text: 'Pretty vector drawing.'". The code "center: &ORIGIN {x: 73, y: 129}; radius: 7" is also present on the right side of the diagram.

```
center:  
  x: 73  
  y: 129  
radius: 7
```

```
finish:  
  x: 89  
  y: 102
```

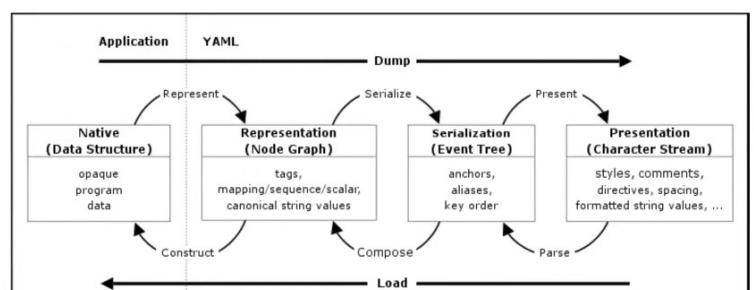
```
start:  
  x: 73  
  y: 129
```

```
color: 16772795  
start:  
  x: 73  
  y: 129  
text: "Pretty vector drawing."  
center:  
  x: 73  
  y: 129  
radius: 7
```

```
start:  
  x: 89  
  y: 102
```

```
color: 0xFFEEBB  
text: Pretty vector drawing.
```

## YAML - processing



## 12. Přednáška

# MULTIMEDIÁLNÍ FORMÁTY

## Grafické formáty

→ **rastr** (pixely) vs. **vektor** (škálovatelné)

## Vektorová grafika

→ 2D, body, cesty (křivky, čáry, polygony)

→ **SVG** – Scalable Vector Graphics (hostitelský formát XML, viz XML) – grafy, plot  
+ možnost embedding v html

## Rastrová grafika

- rozlišení: počet řádků a sloupců (2560x1440)

- celkové množství pixelů (8 Mpx)

- pixelová jemnost = 1200 na palec

- **barvy**: monochromatické, stupně šedi, paleta barev, plná parva

- **barevný model** (matematická reprezentace barvy)

→ RGB (hlavně pro displeje, jak moc září dané barvy)

→ CMYK (kolik inkoustu barvy přijde na danou část papíru)

- **prostor barev** → sRGB (standard RGB) = **mapování RGB**, Adobe RGB, DCI P3

- *gamut* = všechny zobrazitelné barvy

- *bitová hloubka* – počet barev

- **RGBA** = alfa kanál navíc reprezentující průhlednost

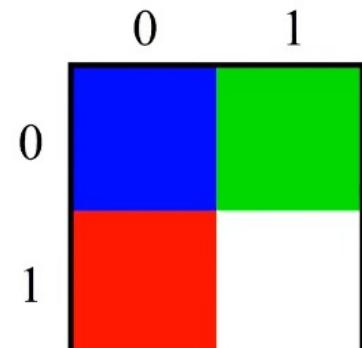
- *dithering* = technika dosahování barvy kombinací všech dostupných hlavních barev

- **DIP** = **device independent pixel** (bez komprese)

formáty:

BMP – **device independent bitmap** (DIB) → obrovské soubory

Offset	Size	Hex value	Value	Description
BMP Header				
DIB Header				
Start of pixel array (bitmap data)				
36h	3	00 00 FF	0 0 255	Red, Pixel (1,0)
39h	3	FF FF FF	255 255 255	White, Pixel (1,1)
3Ch	2	00 00	0 0	Padding for 4 byte alignment (could be a value other than zero)
3Eh	3	FF 00 00	255 0 0	Blue, Pixel (0,0)
41h	3	00 FF 00	0 255 0	Green, Pixel (0,1)
44h	2	00 00	0 0	Padding for 4 byte alignment (could be a value other than zero)



typy komprese:

a) → **bezztrátové komprese**

1) run length kódování → kolik políček dané barvy mám, indexování barev, kde barva končí



1. 4A; 2B; 3C; 1A
2. 3A; 4B; 2C; 1D
3. 2A; 6B; 2D
4. 5B; 5D

1. A4; B6; C9; A10
2. A3; B7; C9; D10
3. A2; B8; D10
4. B5; D10

- 2) blockwise kódování → rozdělení obrázku na **čtvercové bloky** (kde jsou jaké barvy)
- 3) quadtree kódování → **kvandrancy**, dokud nemají bloky stejnou barvu
- 4) **Huffman** kódování → časté hodnoty s nejkratší sekvencí

### Formáty s bezztrátovou kompresí

- **GIF** = Graphics Interchange Format (8 bitů na pixel, LZW komprese) - animace
- **PNG** = Portable network graphics (plný RGB model s alfa kanálem) – bez animací

### b) → ztrátové komprese

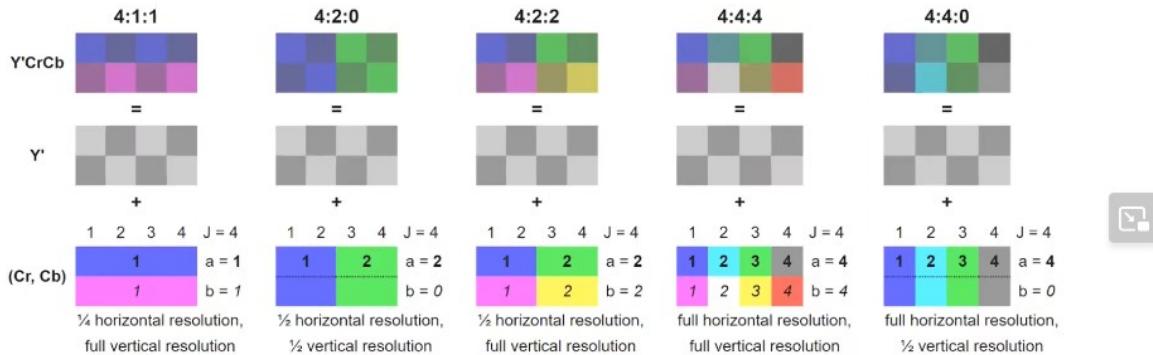
**DCT** – pomocí diskrétní kosinové transformace

- *kvantizace* = vysoké frekvence pryč z obrázku (nižší důležitější pro lidské oko), **kompresie** = jak moc chybí frekvence
- *chroma subsampling* = upravujeme jas (YcbCr), jas, červená, modrá

## Lossy compression - chroma subsampling

J:a:b (e.g. 4:2:2) ~ number of luminance and chrominance samples in a region J pixels wide and 2 pixels high

J - width of the region, usually 4  
a - number of chroma samples in the first row of J pixels  
b - number of changes of chrominance samples between first and second row



**JPEG – komprese:** z RGB do YcbCr, chroma subsampling, pak do 8x8 pixel bloků, DCT, kvantizace, bezztrátová komprese finálních dat  
(progresivní komprese = „postupně se načítá“)

### Rastrové editory – Gimp2, Photoshop

#### Kdy použít?

- **bezztrátové** = obrázky s plochami stejné barvy → malování, screenshoty (PNG, WebP)
- **ztrátové** (lossy) = fotografie (JPEG)

#### RAW formáty

- přímo ze senzoru, obsahuje i intenzitu světla v jednotlivých pixelech
- (dekódování, demozaikování, odstranění vadných pixelů, vyvážení bílé, potlačení šumu, překlad barev, reprodukování tónu včetně mapování a gamma komprese, komprese do nějaké formátu – např. JPG)

## Videoformáty

→ bitmapy za sebou? – hodně omezený postup, ale podporuje ho **R210** (bez komprese)

- MJPEG (motion JPEG)

→ lepší přístup (**předpovídání vnitřku obrázku**):

macroblocks (bloky se zkomprimují), pohybové vektory (pozice macroblock v jednom obrázku založená na pozici z jiného obrázku)

*starší:*

**H261** → staré videotelefonování (pouze 2 rozlišení)

**MPEG** → až 100 Mbps bitrate, Video CD

- různé typy snímků: →

**MPEG-2** → DVD video, **interlaced video** (oddělené řádky)

**H.263** (MMS)

*moderní:*

**MPEG-4** → přesun celého obrázku, nejen makrobloků

(lepší manipulace s jednotlivými pixely), kódování videa

MPEG4

**MPEG-H** – lepší kódování **HEVC**, lepší než **AVC o 50 %**

**MPEG JSOU PLACENÉ! (certifikát)**

**VP8/9** (otevřený videoformát, YouTube používá 9. verzi)

**AV1** – založený na VP9, např. na 8k videa na YouTube

**H.266** – uzavřený, 360 stupňové video

*Rastrová grafika založena na videoformátech*

→ WebP (lepší než GIF, PNG a JPG)

→ HEIC, AVIF (telefony)

Various frame types

- I-frame, intra-frame, keyframe
  - can be rendered independently of others
  - GOP - group of pictures
    - number of frames between two I-frames
    - typically 15-18
- P-frame, predicted frame
  - only difference from the previous frame
  - exploits temporal redundancy
- B-frame, bidirectional / bipredictive frame
  - in addition to P-frame uses also difference from the following frame
- D-frame
  - specific to MPEG-1
  - from low quality DCT

## Audio formáty

**PCM** = pulce-code manipulation

→ **digitální zpracování analogové amplitudy** pomocí regulárních intervalů  
**(sampling rate, frekvence --- 48 kHz)**

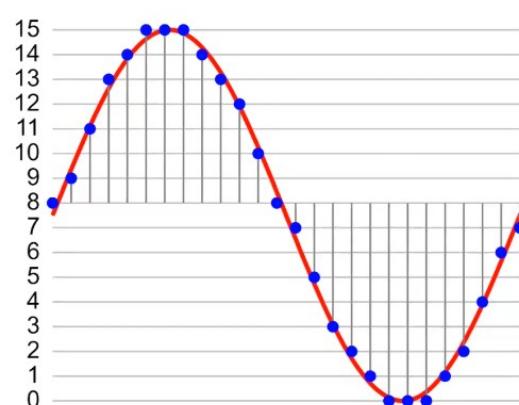
→ **kvantizovaná** na nejbližší hodnotu v oblasti digitálních „kroků“  
**(bit depth, sample size --- 8-bit, ...)**

**LPCM**

- lineárně uniformní kvantizační kroky PCM

**Formáty** → **1.** Wav, CD Audio (bezztrátové)

Frequency	Bit depth	Channels	Bitrate (Kbps)	1 minute size
11 025 Hz	16	1	176.4	1 MB
44 100 Hz	16	2	1 411.2	8.7 MB
96 000 Hz	24	2	4 608	34.5 MB



## **2. FLAC** → 50/70% hodnota komprese (**aproximace vlny pomocí funkce + na rozdělení kanálů**)

(ztrátové formáty):

**3. MP3** → odstraňuje zvuky považované za neslyšitelné běžných posluchačem

**4. AAC** → nástupce MP3, více flexibilní (více typů komprese a její efektivnosti)

**5. OPUS** → otevřený, lepší co se týče kvality než MP3 a AAC (Whatsapp hlasové zprávy)

### **Multimediální formáty**

→ videa se zvukem a titulky (např.)

→ přidání **metadat**

→ multimediální kontejnery (vpravo)

Simple containers

- JPEG, PNG, WAV
- TIFF - images and metadata

Flexible containers - patented, licensed

- AVI - Audio Video Interleave - .avi
- MPEG program stream - .ps .mpg .mpeg
- MPEG-2 transport stream - .ts .m2t
- MP4 - .mp4

Flexible containers - open, royalty-free

- Matroska - .mkv
  - its subset WebM - .webm
- Ogg - .ogg

### **Tiskové formáty**

→ PostScript = programovací jazyk pro tiskárny (vlastní typ media – application/postscript), dnes s bitmap preview

→ **PDF = Portable Document Format**

--- založené na PostScriptu, podpora šifrování  
→ PDF/A = archivní PDF (zakazuje nechtěné funkce), PDF-X (tisk), PDF/VT (invoices), PDF/UA (přístupnost na čtečkách), PDF/E (3d objekty) ---- vše jsou „pod-rodiny“ PDF

- ai = **Adobe Illustrator Artwork**

## 13. Přednáška

### Formáty pro textové dokumenty

- **plain text** (základní text) > má aspoň základní strukturu jako konce řádků

- **formátovaný text** > podtrhování

*starší:*

- **Rich text**

[RFC 1341](#), 1992

- Extremely simple, yet extensible syntax
  - formatting commands between < and >
  - command no more than 40 characters
  - may be preceded by /, making them negations
  - balanced formatting commands and negations, with 3 exceptions
    - <lt>, <nl>, <np>
- Extremely limited capabilities
- Compatibility with SGML

US-ASCII encoding, can be explicitly switched

Media type: text/richtext

```
<b>Now</b> is the time for  
<i>all</i> good men  
<s>(and <lt>women</lt></s>) to  
<ignoreme></ignoreme> come  
  
to the aid of their  
<nl>  
beloved <nl><nl>country. <comment> Stupid  
quote! </comment> -- the end
```

```
Now is the time for all good men (and <women>) to  
come to the aid of their  
beloved  
  
country. -- the end
```

- **Enriched text** = vylepšení Rich text, nabízí syntaktické zkratky (standard pro emaily, byla to ideální alternativa pro HTML, které není moc čitelné a v té době nešlo filtrovat)

**RTF** – Rich Text Format = inspirován TeX, uzavřený

**602** = Československý formát (1988), vlastní formát, z ASCII tabulky

*moderní:*

**HTML** → spíš pro internetovou komunikaci, dříve W3C, dnes WHATWG, markup = **odlišitelný** text vizuálně, **publishing formát**

### **Markdown**

- **markup jazyk**
- **writing formát**
- podpora pro kusy kódu (back-ticky)
- dialekty = různé typy, jak formátovat
- CommonMark
- použití: **readme.md** soubory

# A First Level Header

## A Second Level Header

Now is the time for all good men to come to  
the aid of their country. This is just a  
regular paragraph.

<h1>A First Level Header</h1>

<h2>A Second Level Header</h2>

<p>

Now is the time for all good men to come to  
the aid of their country. This is just a  
regular paragraph.

</p>

1. Red  
2. Green  
3. Blue

some text

<ol>

<li>Red</li>  
<li>Green</li>  
<li>Blue</li>

</ol>

1. Red  
2. Green  
3. Blue

<p>some text</p>

<ol>  
<li>Red</li>  
<li>Green</li>  
<li>Blue</li>

</ol>

some text

CommonMark - fenced code block, info string - "

```
...  
def foo(x)  
  return 3  
end  
...  
    <pre><code>def foo(x)  
      return 3  
    end  
</code></pre>  
  
    <pre><code class="language-ruby">def  
foo(x)  
  return 3  
end  
</code></pre>
```

info string  
usually  
language id

1. Red  
2. Green  
3. Blue

some text

1. Red  
1. Green  
1. Blue

some text

1. Red  
1. Green  
1. Blue

some text

(dodatek: Jekyll --- GitHub Pages, pages v Markdownu, publikovány jako HTML, hlavičky v YAML)

**WikiText** – nemá formální specifikaci, formátování textu na wiki stránkách

```
= Heading 1 =
== Heading 2 ==
=== Heading 3 ===
==== Heading 4 ====
===== Heading 5 =====
===== Heading 6 =====

''italics'', ''bold'', and ''''both''''
```

# Heading 1

## Heading 2

Heading 3  
Heading 4  
Heading 5  
Heading 6

*italics*, **bold**, and **both**

## TeX

→ „hezky vypadající texty“

- otevřený formát, ideální pro vědecké texty (dobrá manipulace s vzorcí a celkově textem)

## LaTeX

→ balíček TeX maker (lepší zaměření na samotný text než na formátování)

- výsledkem je **PDF**

- buď knihovna lokálně či online přes **Overleaf**

- sekce, subsekce (\* odstraním číslo)

- kódování lze nastavit

- 2 nové řádky = odstavec

- podpora seznamů (číslovaných i nečíslovaných)

- citace a reference: **BibTeX**

Ideální: **JEDNA VĚTA, JEDEN ŘÁDEK**

```
\documentclass{article}
\usepackage{amsmath}

\begin{document}
Hello world!
\[
\binom{n}{k} = \frac{n!}{k!(n-k)!}
\]
\end{document}
```

Hello world!

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Commands start with backslashes  
take parameters in [] and {}

\documentclass{class}

- chooses template/style of the document
  - article - no chapters
  - report - with chapters
  - ...

\begin{environment} and \end{environment}

- Marks the start and end of a certain environment, e.g.
  - document
  - definition
  - ...

```
\documentclass{article}
\begin{document}
First document. This is a simple example, with no
extra parameters or packages included.
\end{document}
```

First document. This is a simple example, with no extra parameters or packages included.

LaTeX packages provide functionalities

[CTAN: Comprehensive TeX Archive Network](#)

- Currently 6000+ packages

Packages are defined in preamble

- using \usepackage[options]{package}
- after \documentclass{}
- before \begin{document}

**balíčky:**

```
\documentclass{article}
\usepackage{amsmath}
```

```
\begin{document}
Hello world!
\[
\binom{n}{k} = \frac{n!}{k!(n-k)!}
\]
\end{document}
```



\[ - begin{math mode}



\] - end{math mode}

E.g. package amsmath provides the \binom and \frac commands

**V textu lze:**

- exact match, filtry, wildcards, regulární výrazy
- hledání dokumentů v databázi: invertovaný index (hledání slov, bool, fuzzy search)
- **Dbpedia Spotlight** = hledání pojmenovaných entit v textu
  - RDFa (anotace s RDF)