University of Waterloo E-Thesis Template for LaTeX

by

Pat Neugraad

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Zoology

Waterloo, Ontario, Canada, 2017

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Bruce Bruce Professor, Dept. of Philosophy of Zoology, University of Wallamaloo
Supervisor(s):	Doris Johnson Professor, Dept. of Zoology, University of Waterloo Andrea Anaconda Professor Emeritus, Dept. of Zoology, University of Waterloo
Internal Member:	Pamela Python Professor, Dept. of Zoology, University of Waterloo
Internal-External Member:	Deepa Thotta Professor, Dept. of Philosophy, University of Waterloo
Other Member(s):	Leeping Fang Professor, Dept. of Fine Art, University of Waterloo

Abstract

This is the abstract.

Vulputate minim vel consequat praesent at vel iusto et, ex delenit, esse euismod luptatum augue ut sit et eu vel augue autem feugiat, quis ad dolore. Nulla vel, laoreet lobortis te commodo elit qui aliquam enim ex iriure ea ullamcorper nostrud lorem, lorem laoreet eu ex ut vel in zzril wisi quis. Nisl in autem praesent dignissim, sit vel aliquam at te, vero dolor molestie consequat.

Tation iriure sed wisi feugait odio dolore illum duis in accumsan velit illum consequat consequat ipsum molestie duis duis ut ullamcorper. Duis exerci odio blandit vero dolore eros odio amet et nisl in nostrud consequat iusto eum suscipit autem vero. Iusto dolore exerci, ut erat ex, magna in facilisis duis amet feugait augue accumsan zzril delenit aliquip dignissim at. Nisl molestie nibh, vulputate feugait nibh luptatum ea delenit nostrud dolore minim veniam odio volutpat delenit nulla accumsan eum vero ullamcorper eum. Augue velit veniam, dolor, exerci ea feugiat nulla molestie, veniam nonummy nulla dolore tincidunt, consectetuer dolore nulla ipsum commodo.

At nostrud lorem, lorem laoreet eu ex ut vel in zzril wisi. Suscipit consequat in autem praesent dignissim, sit vel aliquam at te, vero dolor molestie consequat eros tation facilisi diam dolor. Odio luptatum dolor in facilisis et facilisi et adipiscing suscipit eu iusto praesent enim, euismod consectetuer feugait duis. Odio veniam et iriure ad qui nonummy aliquip at qui augue quis vel diam, nulla. Autem exerci tation iusto, hendrerit et, tation esse consequat ut velit te dignissim eu esse eros facilisis lobortis, lobortis hendrerit esse dignissim nisl. Nibh nulla minim vel consequat praesent at vel iusto et, ex delenit, esse euismod luptatum.

Ut eum vero ullamcorper eum ad velit veniam, dolor, exerci ea feugiat nulla molestie, veniam nonummy nulla. Elit tincidunt, consectetuer dolore nulla ipsum commodo, ut, at qui blandit suscipit accumsan feugiat vel praesent. In dolor, ea elit suscipit nisl blandit hendrerit zzril. Sit enim, et dolore blandit illum enim duis feugiat velit consequat iriure sed wisi feugait odio dolore illum duis. Et accumsan velit illum consequat consequat ipsum molestie duis duis ut ullamcorper nulla exerci odio blandit vero dolore eros odio amet et.

In augue quis vel diam, nulla dolore exerci tation iusto, hendrerit et, tation esse consequat ut velit. Duis dignissim eu esse eros facilisis lobortis, lobortis hendrerit esse dignissim nisl illum nulla minim vel consequat praesent at vel iusto et, ex delenit, esse euismod. Nulla augue ut sit et eu vel augue autem feugiat, quis ad dolore te vel, laoreet lobortis te commodo elit qui aliquam enim ex iriure. Ut ullamcorper nostrud lorem, lorem laoreet eu ex ut vel in zzril wisi quis consequat in autem praesent dignissim, sit vel. Dolore at te, vero

dolor molestie consequat eros tation facilisi diam. Feugait augue luptatum dolor in facilisis et facilisi et adipiscing suscipit eu iusto praesent enim, euismod consectetuer feugait duis vulputate veniam et.

Ad eros odio amet et nisl in nostrud consequat iusto eum suscipit autem vero enim dolore exerci, ut. Esse ex, magna in facilisis duis amet feugait augue accumsan zzril. Lobortis aliquip dignissim at, in molestie nibh, vulputate feugait nibh luptatum ea delenit nostrud dolore minim veniam odio. Euismod delenit nulla accumsan eum vero ullamcorper eum ad velit veniam. Quis, exerci ea feugiat nulla molestie, veniam nonummy nulla. Elit tincidunt, consectetuer dolore nulla ipsum commodo, ut, at qui blandit suscipit accumsan feugiat vel praesent.

Dolor zzril wisi quis consequat in autem praesent dignissim, sit vel aliquam at te, vero. Duis molestie consequat eros tation facilisi diam dolor augue. Dolore dolor in facilisis et facilisi et adipiscing suscipit eu iusto praesent enim, euismod consectetuer feugait duis vulputate.

Acknowledgements

I would like to thank all the little people who made this thesis possible.

Dedication

This is dedicated to the one I love.

Table of Contents

List of Tables

List of Figures

Abbreviations

AAAAZ American Association of Amateur Astronomers and Zoologists 1

Nomenclature

dingledorf	A person of supposed average intelligence who makes incredibly brainless misjudgments 1

See equation **??** on page **??**.[1]

0.1 Some Meaningless Stuff

The credo of the American Association of Amateur Astronomers and Zoologists (AAAAZ) was, for several years, several paragraphs of gibberish, until the dingledorf responsible for the AAAAZ Web site realized his mistake:

"Velit dolor illum facilisis zzril ipsum, augue odio, accumsan ea augue molestie lobortis zzril laoreet ex ad, adipiscing nulla. Veniam dolore, vel te in dolor te, feugait dolore ex vel erat duis nostrud diam commodo ad eu in consequat esse in ut wisi. Consectetuer dolore feugiat wisi eum dignissim tincidunt vel, nostrud, at vulputate eum euismod, diam minim eros consequat lorem aliquam et ad. Feugait illum sit suscipit ut, tation in dolore euismod et iusto nulla amet wisi odio quis nisl feugiat adipiscing luptatum minim nisl, quis, erat, dolore. Elit quis sit dolor veniam blandit ullamcorper ex, vero nonummy, duis exerci delenit ullamcorper at feugiat ullamcorper, ullamcorper elit vulputate iusto esse luptatum duis autem. Nulla nulla qui, te praesent et at nisl ut in consequat blandit vel augue ut.

Illum suscipit delenit commodo augue exerci magna veniam hendrerit dignissim duis ut feugait amet dolor dolor suscipit iriure veniam. Vel quis enim vulputate nulla facilisis volutpat vel in, suscipit facilisis dolore ut veniam, duis facilisi wisi nulla aliquip vero praesent nibh molestie consectetuer nulla. Wisi nibh exerci hendrerit consequat, nostrud lobortis ut praesent dignissim tincidunt enim eum accumsan. Lorem, nonummy duis iriure autem feugait praesent, duis, accumsan tation enim facilisi qui te dolore magna velit, iusto esse eu, zzril. Feugiat enim zzril, te vel illum, lobortis ut tation, elit luptatum ipsum, aliquam dolor sed. Ex consectetuer aliquip in, tation delenit dignissim accumsan consequat, vero, et ad eu velit ut duis ea ea odio.

Vero qui, te praesent et at nisl ut in consequat blandit vel augue ut dolor illum facilisis zzril ipsum. Exerci odio, accumsan ea augue molestie lobortis zzril laoreet ex ad, adipiscing nulla, et dolore, vel te in dolor te, feugait dolore ex vel erat duis. Ut diam commodo ad eu in consequat esse in ut wisi aliquip dolore feugiat wisi eum dignissim tincidunt vel, nostrud. Ut vulputate eum euismod, diam minim eros consequat lorem aliquam et ad luptatum illum sit suscipit ut, tation in dolore euismod et iusto nulla. Iusto wisi odio quis nisl feugiat adipiscing luptatum minim. Illum, quis, erat, dolore qui quis sit dolor veniam blandit ullamcorper ex, vero nonummy, duis exerci delenit ullamcorper at feugiat. Et, ullamcorper elit vulputate iusto esse luptatum duis autem esse nulla qui.

[1] A famous equation.

Praesent dolore et, delenit, laoreet dolore sed eros hendrerit consequat lobortis. Dolor nulla suscipit delenit commodo augue exerci magna veniam hendrerit dignissim duis ut feugait amet. Ad dolor suscipit iriure veniam blandit quis enim vulputate nulla facilisis volutpat vel in. Erat facilisis dolore ut veniam, duis facilisi wisi nulla aliquip vero praesent nibh molestie consectetuer nulla, iriure nibh exerci hendrerit. Vel, nostrud lobortis ut praesent dignissim tincidunt enim eum accumsan ea, nonummy duis. Ad autem feugait praesent, duis, accumsan tation enim facilisi qui te dolore magna velit, iusto esse eu, zzril vel enim zzril, te. Nisl illum, lobortis ut tation, elit luptatum ipsum, aliquam dolor sed minim consectetuer aliquip.

Tation exerci delenit ullamcorper at feugiat ullamcorper, ullamcorper elit vulputate iusto esse luptatum duis autem esse nulla qui. Volutpat praesent et at nisl ut in consequat blandit vel augue ut dolor illum facilisis zzril ipsum, augue odio, accumsan ea augue molestie lobortis zzril laoreet. Ex duis, te velit illum odio, nisl qui consequat aliquip qui blandit hendrerit. Ea dolor nonummy ullamcorper nulla lorem tation laoreet in ea, ullamcorper vel consequat zzril delenit quis dignissim, vulputate tincidunt ut.”

Chapter 1

Development of a Python library for programmatic exploration and comparison of organism Genome Properties

Introduction text..

1.1 Parsing the Genome Properties Database

The Genome Properties database consists of a series of flat files, whose individual property records are not indexed or connected. In all use cases, Pygenprop requires the information found within the Genome Properties database to perform its job. Before this information can be used by the library, it must be loaded into main memory. The intent of Pygenprop's parser is to read the database files from disk, load them into main memory, build connections between records found within and present the information contained to the rest of the library.

1.1.1 Overview of the Genome Properties flat file database and associated file formats

The Genome Properties database currently consists of a series of flat files which are hosted inside a Github Repository (see URL). Information about both public and non-public properties are hosted under this repository's **data** folder. Each property is assigned a single file folder which contains three files. A **DESC** file, which contains information about the property; a **status** file which contains information onto whether the property is public or has been manually curated; and a **FASTA** file, for properties whose steps are supported by InterProScan signatures, which contain representative protein sequences for each step of the property. In addition to the per-property folders contained within the repository's **data** folder, there is also a Genome Properties release file located in the **flatfiles** folder which also contains Genome Properties information. Specifically, this file, called **genomeProperties.txt**, is a concatenation of the **DESC** files for all public properties found in the repositories **data** folder and is created with each release of the Genome Properties database on Github. Below is simplified a folder structure for the Genome Properties Github repository.

```
code/ - # Contains the Genome Properties Perl library.
data/ - # Data about both public and private properties
   GenProp0001/
      DESC - # Detailed property information
      FASTA - # Example sequences of proteins that carry out each step of the property
      status - # Contains public and manual curation statuses
   GenProp0002/
       DESC
       FASTA
       status
flatfiles/
    genomeProperties.txt
```

Pygenprop contains a parser for parsing both the **DESC** files of single singular property folders and the concatenated **genomeProperties.txt** file. The format of each **DESC** file is very similar to the Stockholm sequence alignment format used by both the Pfam and Rfam databases [1, 5] and as such the format consists of key value pairs. However, since these files use different keys than Stockholm a custom parser had to be developed. It is of note that the Genome Properties database format wraps every eighty characters. Thus, some key types which contain long sentences will be repeated for multiple lines. Below is an example **DESC** file and a summary of key types can be found in Table 1.1.

```
AC  GenProp0145
DE  Histidine degradation to glutamate
TP  PATHWAY
AU  Haft DH
TH  2
RN  [1]
RM  2203753
RT  Nucleotide sequence of the gene encoding the repressor for the
RT  histidine utilization genes of Pseudomonas putida.
RA  Allison SL, Phillips AT;
RL  J Bacteriol. 1990;172:5470-5476.
RN  [2]
RM  25559274
RT  Structure of N-formimino-L-glutamate iminohydrolase from Pseudomonas
RT  aeruginosa.
RA  Fedorov AA, Mart-Arbona R, Nemmara VV, Hitchcock D, Fedorov EV, Almo SC,
RA  Raushel FM;
RL  Biochemistry. 2015;54(3):890-7.
DC  Histidine Catabolism
DR  IUBMB; AminoAcid; His3;
DC  Histidine Metabolism
DR  KEGG; map00340;
DC  L-histidine degradation II
DR  MetaCyc; PWY-5028;
CC  This pathway is involved in histidine utilization system (hut). HutP is
CC  the first gene in the hut operon encoding the hutHUIG operator and a
CC  positive regulator of the operon, activated allostatically in the
CC  presence of L-histidine. HutC represses histidine utilization by binding
CC  the regulatory sites for hutHUIG and hutF [1]. There are multiple
CC  variations in the histidine degradation pathway, including two possible
CC  routes for the first step (either via histidine transaminase, or as in
CC  this pathway, via histidine ammonia-lyase/histidase). L-histidine is
CC  first converted to urocanate by hutH (histidine ammonia-lyase), which is
CC  then converted to 4-imidazolone-5-propionate by hutU (urocanate
CC  hydratase), and finally hydrolysed to N-formimino-L-glutamate by hutI
CC  (imidazolonepropionate amidohydrolase). From here there are three
CC  potential paths to glutamate. This property refers to the two-step
CC  process found in some bacteria where N-formimino-L-glutamate is first
CC  converted to N-formyl-l-glutamate by hutF (formimidoylglutamate
CC  deiminase) and then hydrolyzed to L-glutamate by hutG
CC  (N-formyl-l-glutamate deformylase)[2].
**  Evidence for steps 4 and 5 is the same.
--
SN  1
ID  Histidine ammonia-lyase (hutH)
DN  Histidine ammonia-lyase/hutH (EC 4.3.1.3)
RQ  1
EV  IPR005921; TIGR01225; sufficient;
TG  GO:0006548;
--
SN  2
ID  Urocanate hydratase (hutU)
DN  Urocanate hydratase/hutU (EC 4.2.1.49)
RQ  1
EV  IPR023637; TIGR01228; sufficient;
TG  GO:0006548;
--
SN  3
ID  Imidazolonepropionase (hutI)
DN  Imidazolonepropionase/hutI (EC 3.5.2.7)
RQ  1
EV  IPR005920; TIGR01224; sufficient;
TG  GO:0006548;
--
SN  4
ID  Formimidoylglutamate deiminase/formiminoglutamase/glu-formyltransferase
DN  Formimidoylglutamate deiminase/hutF (EC 3.5.3.13)
RQ  1
EV  IPR005923; TIGR01227; sufficient;
TG  GO:0006548;
EV  IPR010252; TIGR02022; sufficient;
TG  GO:0006548;
EV  IPR004227; TIGR02024; sufficient;
```

```
TG  GO:0006548;
--
SN  5
ID  Formylglutamate deformylase/formiminoglutamase/glu-formyltransferase
DN  N-formylglutamate deformylase/hutG (EC 3.5.1.68)
RQ  1
EV  IPR005923; TIGR01227; sufficient;
TG  GO:0006548;
EV  IPR010247; TIGR02017; sufficient;
TG  GO:0006548;
EV  IPR004227; TIGR02024; sufficient;
TG  GO:0006548;
--
SN  6
ID  Histidine utilization repressor (hutC)
DN  Histidine utilization repressor/hutC
RQ  0
EV  IPR010248; TIGR02018; sufficient;
//
```

Table 1.1: Genome Properties DESC files use a variety of keys to provide information about a single property. Note that this table is copied form the Genome Properties database documentation (see https://genome-properties.readthedocs.io/en/latest/flatfile.html#desc-file).

Key	**Information Type**
AC	Accession ID
DE	Description/name of Genome Property
TP	Type
AU	Author
TH	Threshold
RN	Reference number
RM	PMID of reference
RT	Reference title
RA	Reference author
RL	Reference citation
DC	Database title

Table 1.1 continued from previous page

Key	**Information Type**
DR	Database link
PN	Parent accession ID
CC	Property description
**	Private notes
	Separator
SN	Step number
ID	Step ID
DN	Step display name (includes EC number if available)
RQ	Required step
EV	Evidence (includes whether sufficient)
TG	Gene Ontology (GO) ID
//	End

1.1.2 Parser Implementation

Pygenprop's Genome Properties flat file parser can parse both single property **DESC** files and **genomeProperties.txt** database release files which contain information about multiple properties. It reads these files one line at a time to decrease memory usage, allowing for compatibility with low memory machines and increases in database size. While loading line by line, lines for each property are loaded into a Python list as they are encountered. Once a list for a single property is full, the key types which can take up multiple lines, such as property descriptions (see Table 1.1 and example file above), are collapsed to single key value pairs. These collapsed key-value pairs are then iterated and the data inside are used to create a series of in-memory objects representing the property. As individual property objects are created they are added to a list. Once parsing is completed, the parser places this list in a Genome Property Tree object which represents the connections in the database's DAG structure. This object is then returned from the parser.

1.1.3 Parser Performance

Pygenprop's Genome Properties flat file parser was found to be able to parse single **DESC** files in 415 s 5.59 s on average and the latest release of the entire Genome Properties database (**genomeProperties.txt** of release 2.0) in 242 ms 4.81 ms (using a Macbook

Pro 13-inch, Late 2013 with an Intel Intel Core i5 2.4 GHz processor). Since most applications of the parser will involve only parsing the database once, this speed was determined to be sufficient. If a greater speed is required, for example if the genome properties database grows greatly in size, the parser could be sped up by using software such as Cython [2] or Numba [10] to transpile the existing Python code to C [7]. Alternatively, the parser could be rewritten in C or C++ [6] from scratch and integrated into the existing Python code via CPython's C extension interface [14]. If the machine that Pygenprop is running on is I/O bound, other solution may be required such as storing the Genome Properties database in a Random-access memory (RAM) disk or on a Solid-state drive (SSD).

1.2 Development of an object oriented class framework for the representation of the Genome Properties database

As discussed in the previous chapter, the Genome Properties database consists of series of interdependent genome properties representing both metabolic and structural features of cells. Some properties are used as evidence of others forming parent child relationships between properties and an overall rooted directed acyclic graph structure (DAG). After parsing the Genome Properties database, Pygenprop instantiates a series of objects representing that information contained within the database (see Table 1.2, Fig. 1.2 and Fig. 1.1). These objects are connected to each other in linked list fashion where objects point to each othe. These connections are doubly linked facilitating climbing both up and down the genome properties DAG and between genome property, step, functional element and evidence objects (Fig. 1.2 and Fig. 1.1). Individual methods and attributes of these objects can be used in software applications or used interactively in Jupyter Notebooks [8]. The below subsections detail the Genome Properties database classes and how they can be used.

Table 1.2: A summary of the object types used to represent the Genome Properties database.

Object Type	**Description**
Tree	Encapsulates a DAG of genome property objects
Genome Property	Represents an individual genome property
Literature Reference	Represents an article discussing a genome property

Table 1.2 continued from previous page

Object Type	**Description**
Database Reference	Represents a record in an external pathways database which is equivalent to a genome property
Step	Represents a step supporting the existence of a genome property
Functional Element	Represents a functional element supporting the existence of a step
Evidence	Represents an evidence supporting the existence of a functional element

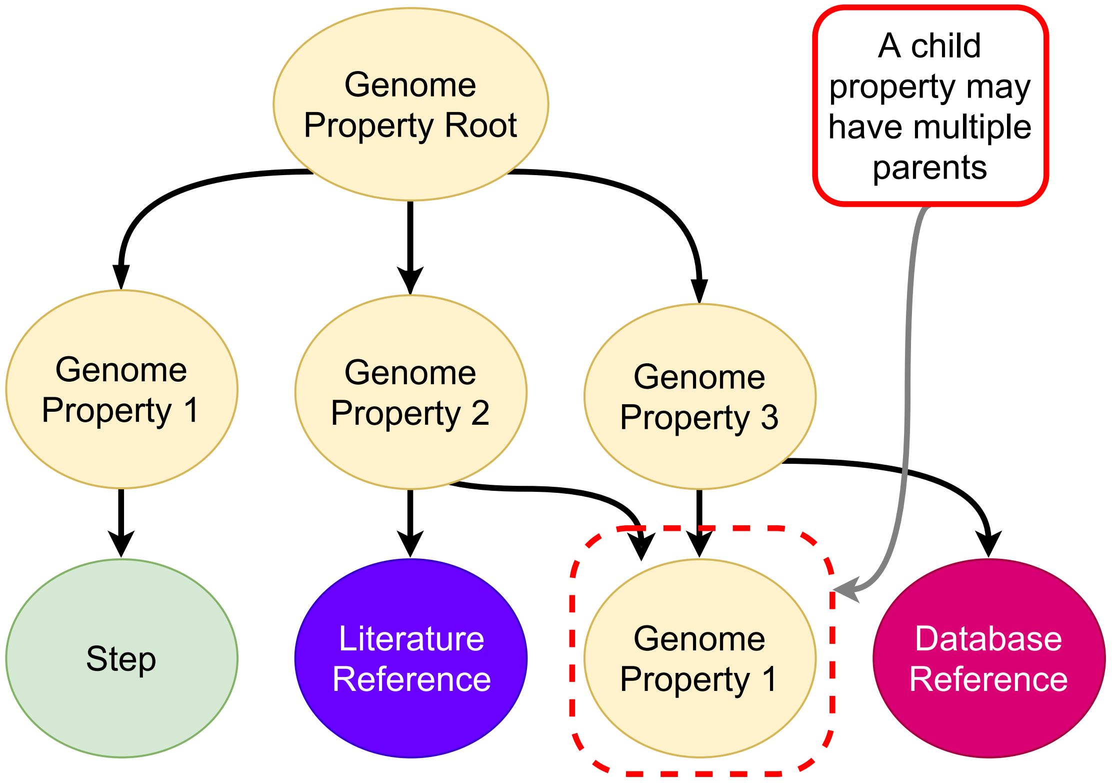

Figure 1.1: Some property objects are the children of others. Database reference, literature reference and step objects are children of property objects. Figure is from [3].

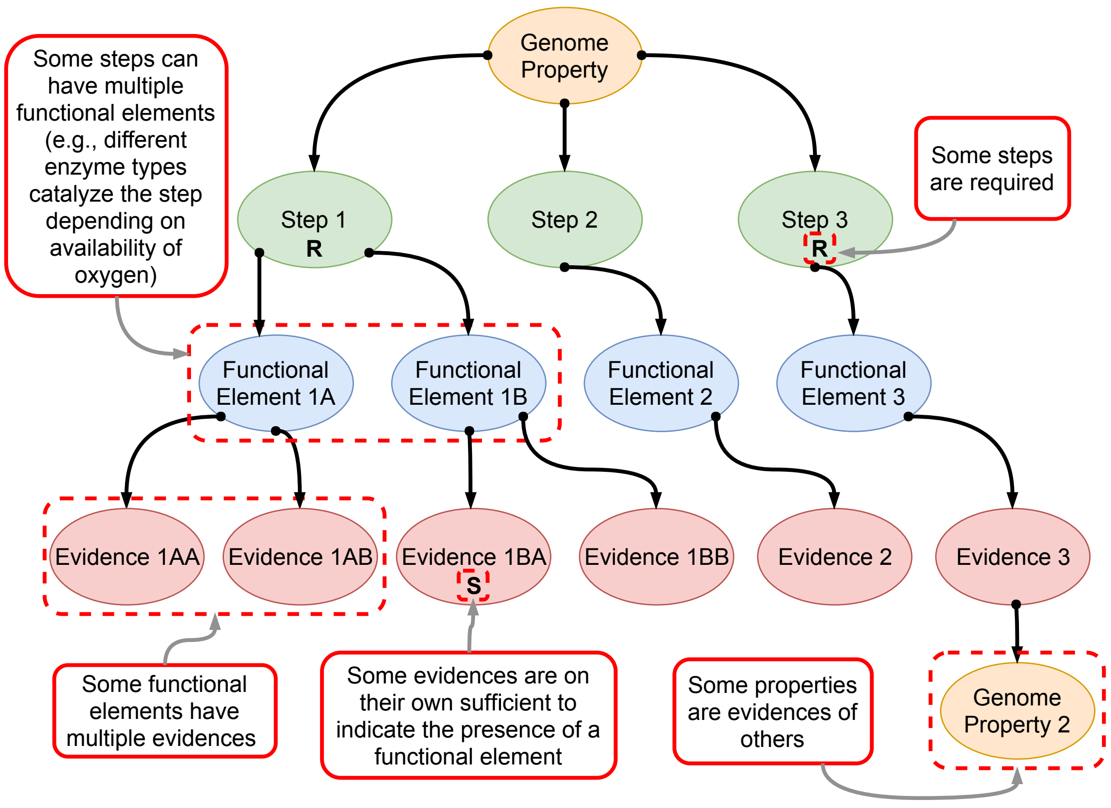
Genome Property
Some steps can have multiple functional elements (e.g., different enzyme types catalyze the step depending on availability of oxygen)
Step 1 R
Step 2
Step 3 R
Some steps are required
Functional Element 1A
Functional Element 1B
Functional Element 2
Functional Element 3
Evidence 1AA
Evidence 1AB
Evidence 1BA S
Evidence 1BB
Evidence 2
Evidence 3
Some functional elements have multiple evidences
Some evidences are on their own sufficient to indicate the presence of a functional element
Some properties are evidences of others
Genome Property 2

Figure 1.2: Each property is supported by step, functional element, and evidence objects. Figure is from [3].

1.2.1 The Genome Property Class

The genome property class creates a blueprint for objects which represent individual genome properties. Instantiated objects possess methods, properties (attributes whose return value is generated by a function), and attributes which represents data about the property contained in the property **DESC** file. Information about property steps, database references and literature references have been abstracted into their own classes. A summary of the methods, properties and attributes of genome property objects can be seem in Table 1.3 and example code below.

Table 1.3: A list of methods, properties and attributes of genome property objects.

Name	**Type**	**Description**
required_steps	Property	Return a list of step objects representing steps which are required to support the existence of the property
child_genome _property _identifiers	Property	Return a list of the genome property identifiers of child genome properties which are used as step evidences for the property
to_json	Method	Serialize the property to a JSON string
databases	Attribute	A list of database objects representing external database references to the property
references	Attribute	A list of literature reference objects representing external articles discussing the property
private_notes	Attribute	Private internal notes about the property
tree	Attribute	The genome property tree for to which the property belongs
description	Attribute	A complete description for the property
threshold	Attribute	The minimum number of required steps for to which must be assigned YES in order for the property to be assigned PARTIAL rather than NO support during property assignment
type	Attribute	The type of property (e.g. GUILD, CATEGORY, PATHWAY, etc.)
steps	Attribute	A list of step objects representing all steps that can support the existence of the property (including non-required)
public	Attribute	True if the property is publicly released
children	Attribute	A list of child genome property objects representing properties the are used as step evidences by the property
name	Attribute	The name of the property
id	Attribute	The genome property identifier (e.g. GenPropXXXX)
parents	Attribute	A list of parent genome properties objects representing properties that use the property as step evidences

Example code for using genome property objects

```
property.id
Out: 'GenProp0144'

property.name
Out: 'Chlorophyllide␣a␣biosynthesis␣from␣protoporphyrin␣IX'

property.parents
Out: List of parent property objects

property.children
Out: List of child property objects

property.steps
Out: List of step objects

property.databases
Out: List of database reference objects

property.references
Out: List of literature reference objects
```

1.2.2 The Database Reference Class

The database reference class allows for the creation of objects which map the property to equivalent records in other databases such as KEGG and Metacyc. They are children of genome property objects. A summary of the methods, properties and attributes of database reference objects can be seem in Table 1.4 and example code below.

Table 1.4: A list of methods, properties and attributes of database reference objects.

Name	**Type**	**Description**
database_name	Attribute	The name of the database in questions (e.g. KEGG)
record_title	Attribute	The name of the record in the external database for which the property is equivalent
record_ids	Attribute	The identifier of the record in the external database for which the property is equivalent

Example code for using database reference objects

```
reference = property.databases[0]

reference.database_name
Out: 'MetaCyc'

reference.record_title
Out: 'Pathway: 3,8-divinyl-chlorophyllide a biosynthesis III'

# Returns a list to handle cases where there are multiple identifiers.
reference.record_ids[0]
Out: 'PWY-7159'
```

1.2.3 The Literature Reference Class

The literature reference class lays out the foundation for objects which represent specific articles which support the existence of the property. They are children of genome property objects. A summary of the methods, properties and attributes of literature reference objects can be seem in Table 1.5 and example code below.

Table 1.5: A list of methods, properties and attributes of literature reference objects.

Name	Type	Description
number	Attribute	The number of the reference
pubmed_id	Attribute	The PUBMED identifier of the reference
title	Attribute	The title of the literature reference for the property
authors	Attribute	The authors of the literature reference for the property
citation	Attribute	A citation for the literature reference for the property

Example code for using literature reference objects

```
reference = property.references[0]

reference.pubmed_id
```

```
Out: '17370354'

reference.title
Out: 'Recent␣advances␣in␣chlorophyll␣biosynthesis.'

reference.citation
Out: 'Photosynth␣Res.␣2006;90(2):173-194.'
```

1.2.4 The Step Class

The step class is used to generate objects representing individual genome property steps. They are children of parent genome properties. They also have functional elements as children. A summary of the methods, properties and attributes of step objects can be seem in Table 1.6 and example code below.

Table 1.6: A list of methods, properties and attributes of step objects.

Name	**Type**	**Description**
name	Property	Return the name of the step
required	Property	Return true if the step is required for assignment of the parent genome property
property _identifiers	Property	Return a list of genome property identifiers of genome properties which are used as evidence for the step
interpro _identifiers	Property	Return a list of InterPro identifiers which are used as evidence for the step (e.g. IPRXXXX)
consortium _identifiers	Property	Return a list of InterPro consortium member database (e.g. PFAM) signature identifiers which are used as evidence for the step (e.g. PFXXXXX)
genome _properties	Property	Return a list of child genome property objects which are used as evidence for the step
number	Attribute	The number of the step
parent	Attribute	The parent genome property of the step
functional _elements	Attribute	A list of functional elements which are used to support the existence a step

Example code for using step objects

```
step = property.steps[0]

step.number
Out: '1'

step.name
Out: 'Magnesium-chelatase␣subunit␣ChlD␣(EC␣6.6.1.1)'

step.required
Out: 'True'

step.interpro_identifiers
Out: 'A␣list␣of␣InterPro␣identifiers␣(e.g.␣IPR011776)'

step.consortium_identifiers
Out: 'A␣list␣of␣consortium␣signature␣identifiers␣(e.g.␣TIGR02031)'

step.functional_elements
Out: 'A␣list␣of␣functional␣element␣objects'
```

1.2.5 The Functional Element Class

The functional element class allows for the instantiation of objects which are placed between step object and evidence objects during parsing. Functional elements are not part of the original genome properties database schema and were added by Pygenprop to take into account for certain steps which can be catalysed by multiple enzyme families. For example, it is common that under anoxic conditions organisms will use a different set of enzymes to catalyze a step in a biochemical pathway due to the lack of oxygen present to support the reaction. This issue of having multiple types of enzymes being able to catalyze a step is an open issue on the Genome Properties database Github (see https://github.com/ebi-pf-team/genome-properties/issues/29). The addition of functional elements is designed to address this issue. A summary of the methods, properties and attributes of functional element objects can be seem in Table 1.7 and example code below.

Table 1.7: A list of methods, properties and attributes of functional element objects.

Name	Type	Description
parent	Attribute	The step object for to which the functional element supports
evidence	Attribute	A list of evidence objects that support the existence of the functional element
name	Attribute	The name of the functional element
id	Attribute	The identifier of the functional element
required	Attribute	True if the functional element is required for assignment of the parent genome property

Example code for using functional element objects

```
element = step.functional_elements[0]

element.id
Out: 'element.id'

element.name
Out: 'Magnesium-chelatase subunit ChlD (EC 6.6.1.1)'

element.required
Out: 'True'

element.evidence
Out: 'A list of evidence objects'
```

1.2.6 The Evidence Class

The evidence class allows for the generations of objects which represent individual pieces of evidence which support the existence of functional elements and in turn genome property steps. Pieces of evidence include the presence of InterPro consortium signatures or support for existence of other genome properties found in an organism's genome. A summary of the methods, properties and attributes of evidence objects can be seem in Table 1.7 and example code below.

Table 1.8: A list of methods, properties and attributes of evidence objects.

Name	**Type**	**Description**
has_genome _property	Property	Return true if the evidence is supported by the existence a genome property
property _identfiers	Property	Return a list of genome property identifiers of genome properties which are used by the evidence
interpro _identifiers	Property	Return a list InterPro identifiers of genome properties which are used by this evidence (e.g. IPRXXXX)
consortium _identifiers	Property	Return a list of InterPro consortium member database (e.g. PFAM) signature identifiers of genome properties which are used by this evidence (e.g. PFXXXXX)
genome _properties	Property	Return a list of child genome property objects which are used by this evidence
parent	Attribute	The parent functional element of this evidence
gene_ontology _terms	Attribute	The GO term identifiers associated with the InterPro identifiers which are used by the evidence
evidence _identifiers	Attribute	A list of both InterPro and signature identifiers used by the evidence
sufficient	Attribute	True if the evidence alone can prove the existence of a functional element

Example code for using evidence objects

```
evidence = element.evidence[0]

evidence.has_genome_property
Out: 'false'

evidence.sufficient
Out: 'true'

evidence.interpro_identifiers
Out: 'A list of InterPro identifiers (e.g. IPR011776)'

evidence.consortium_identifiers
```

```
Out: 'A␣list␣of␣consortium␣signature␣identifiers␣(e.g.␣TIGR02031)'
```

1.2.7 The Genome Properties Tree Class

Genome properties tree objects, as instantiated from the genome properties tree class, represent the rooted DAG structure of entire Genome Properties database. even though the Genome Properties database is actually a rooted DAG, the name 'tree' is used for the class and tree terminology is used in the object's methods for end user convenience. A rooted DAG is not a tree as its branches can merge together unlike those of a true tree. Tree objects contains a Python dictionary of genome property objects indexed by their property identifiers. In addition, individual property objects point to each other using their child and parent (Fig. 1.1 and Table 1.3). These child-parent relationships between property objects are built the genome properties tree object's instantiation. The genome properties tree class allows users to search for specific genome properties, and find root and leaf properties. A summary of the methods, properties and attributes of tree objects can be seem in Table 1.9 and example code below.

Table 1.9: A list of methods, properties and attributes of tree objects.

Name	**Type**	**Description**
build_genome_property_connections	Method	Iterate through every genome property which is a child of the tree; set these property's parent and child attributes to point to child and parent property objects which are also children of the tree. This method connects property objects to create a rooted DAG structure.
to_json	Method	Serialize the property tree to a JSON string
create_metabolism_database_mapping_file	Method	Write a CSV file which maps from genome property identifiers to the identifiers of equivelent records found in KEGG and Metacyc
root	Property	The genome property who has no parent.
leafs	Property	Return a list of genome property objects whose steps are not supported by other genome properties
genome_property_identifiers	Property	Return a list of the genome property identifiers (e.g. GenPropXXXX) for all genome properties within the database
interpro_identifiers	Property	Return a list of InterPro identifiers which are used as evidence for all steps (e.g. IPRXXXX) within the database

Table 1.9 continued from previous page

Name	Type	Description
consortium _identifiers	Property	Return a list of InterPro consortium member database (e.g. PFAM) signature identifiers which are used as evidence for the step (e.g. PFXXXXX) within the database
consortium _identifiers _dataframe	Property	Return the above in the form of a pandas DataFrame
genome _properties _dictionary	Attribute	A dictionary of genome property objects representing all genome properties within by the database; dictionary is keyed by genome property identifier

Example code for using genome property tree objects

```
tree = GenomePropertyTree(*property_object_list)
tree_two = parse_genome_properties_flat_file(genome_properties_file_handle)

len(tree)  # number of properties in the database
Out: 'false'

tree.root
Out: 'The root genome property object'

tree.leafs
Out: 'A list of leaf genome property objects (those with no child properties

for genome_property in tree:  # Properties in the tree can be iterated.
        print(genome_property.id)
Out: 'Prints all genome property identifiers'

tree['GenProp1127'] # The tree can be rapidly searched
Out: 'The genome property object representing GenProp1127.'
```

1.2.8 Performance of Pygenprop's Genome Properties database representation

Pygenprop's representation of the Genome Properties database (Version 2.0), a genome properties tree object and its children, takes only up 11.16 MB of random-access memory. This in contrast to the database's original **genomeProperties.txt** file which takes up only 1.76 MB on disk. The memory usage difference is due the representation of the database as a series of objects and their associated data structures. However, since 11.16 MB is still takes up little memory on a modern machine, more compact data representations for Genome Properties data were not pursued. The size of the database, and its read-only use case, allows for its storage in main memory rather than in an on-disk database such as SQLite [13] or PostgreSQL [12].

Individual genome property objects can be looked up, by property identifier, from within a genome properties tree object within 277 ns 7.91 ns. This speed is due property objects being stored within a Python dictionary. Python dictionaries are implemented a hash tables, allowing for quick look ups [14].

1.3 Assignment of Properties to Organism Genomes

1.3.1 The Assignment Cache class

1.3.2 The Assignment Algorithm

Assignment of steps, functional elements and evidences

Assignment of non-categorical properties

Assignment of categorical properties

1.3.3 Assignment Algorithm Performance

Chapter 2

Observations

This would be a good place for some figures and tables.

Some notes on figures and photographs...

- A well-prepared PDF should be
  1. Of reasonable size, *i.e.* photos cropped and compressed.
  2. Scalable, to allow enlargment of text and drawings.
- Photos must be bit maps, and so are not scaleable by definition. TIFF and BMP are uncompressed formats, while JPEG is compressed. Most photos can be compressed without losing their illustrative value.
- Drawings that you make should be scalable vector graphics, *not* bit maps. Some scalable vector file formats are: EPS, SVG, PNG, WMF. These can all be converted into PNG or PDF, that pdflatex recognizes. Your drawing package probably can export to one of these formats directly. Otherwise, a common procedure is to print-to-file through a Postscript printer driver to create a PS file, then convert that to EPS (encapsulated PS, which has a bounding box to describe its exact size rather than a whole page). Programs such as GSView (a Ghostscript GUI) can create both EPS and PDF from PS files. Appendix A shows how to generate properly sized Matlab plots and save them as PDF.
- It's important to crop your photos and draw your figures to the size that you want to appear in your thesis. Scaling photos with the includegraphics command will cause

loss of resolution. And scaling down drawings may cause any text annotations to become too small.

For more information on LaTeX see the uWaterloo Skills for the Academic Workplace course notes. [1]

The classic book by Leslie Lamport [11], author of LaTeX, is worth a look too, and the many available add-on packages are described by Goossens *et al* [4].

[1] Note that while it is possible to include hyperlinks to external documents, it is not wise to do so, since anything you can't control may change over time. It *would* be appropriate and necessary to provide external links to additional resources for a multimedia "enhanced" thesis. But also note that if the **hyperref** package is not included, as for the print-optimized option in this thesis template, any `\href` commands in your logical document are no longer defined. A work-around employed by this thesis template is to define a dummy `\href` command (which does nothing) in the preamble of the document, before the **hyperref** package is included. The dummy definition is then redifined by the **hyperref** package when it is included.

References

[1] Alex Bateman, Lachlan Coin, Richard Durbin, Robert D Finn, Volker Hollich, Sam Griffiths-Jones, Ajay Khanna, Mhairi Marshall, Simon Moxon, Erik LL Sonnhammer, et al. The pfam protein families database. *Nucleic acids research*, 32(suppl_1):D138–D141, 2004.

[2] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D.S. Seljebotn, and K. Smith. Cython: The best of both worlds. *Computing in Science Engineering*, 13(2):31 –39, 2011.

[3] Lee H Bergstrand, Josh D Neufeld, and Andrew C Doxey. Pygenprop: a python library for programmatic exploration and comparison of organism genome properties. *Bioinformatics*, 2019.

[4] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison-Wesley, Reading, Massachusetts, 1994.

[5] Sam Griffiths-Jones, Alex Bateman, Mhairi Marshall, Ajay Khanna, and Sean R Eddy. Rfam: an rna family database. *Nucleic acids research*, 31(1):439–441, 2003.

[6] ISO. *ISO/IEC 14882:1998: Programming languages — C++*. September 1998. Available in electronic form for online purchase at `http://webstore.ansi.org/` and `http://www.cssinfo.com/`.

[7] Brian W Kernighan and Dennis M Ritchie. *The C programming language*. 2006.

[8] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian E Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica B Hamrick, Jason Grout, Sylvain Corlay, et al. Jupyter notebooks-a publishing format for reproducible computational workflows. In *ELPUB*, pages 87–90, 2016.

[9] Donald Knuth. *The TeXbook*. Addison-Wesley, Reading, Massachusetts, 1986.

[10] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. Numba: A llvm-based python jit compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, page 7. ACM, 2015.

[11] Leslie Lamport. *LaTeX — A Document Preparation System.* Addison-Wesley, Reading, Massachusetts, second edition, 1994.

[12] Bruce Momjian. *PostgreSQL: introduction and concepts*, volume 192. Addison-Wesley New York, 2001.

[13] Mike Owens. *The definitive guide to SQLite.* Apress, 2006.

[14] Guido Van Rossum and Fred L Drake Jr. *Python reference manual.* Centrum voor Wiskunde en Informatica Amsterdam, 1995.

APPENDICES

Appendix A

Matlab Code for Making a PDF Plot

A.1 Using the Graphical User Interface

Properties of Matab plots can be adjusted from the plot window via a graphical interface. Under the Desktop menu in the Figure window, select the Property Editor. You may also want to check the Plot Browser and Figure Palette for more tools. To adjust properties of the axes, look under the Edit menu and select Axes Properties.

To set the figure size and to save as PDF or other file formats, click the Export Setup button in the figure Property Editor.

A.2 From the Command Line

All figure properties can also be manipulated from the command line. Here's an example:

```
x=[0:0.1:pi];
hold on % Plot multiple traces on one figure
plot(x,sin(x))
plot(x,cos(x),'--r')
plot(x,tan(x),'.-g')
title('Some Trig Functions Over 0 to \pi') % Note LaTeX markup!
legend('{\it sin}(x)','{\it cos}(x)','{\it tan}(x)')
hold off
```

```
set(gca,'Ylim',[-3 3]) % Adjust Y limits of "current axes"
set(gcf,'Units','inches') % Set figure size units of "current figure"
set(gcf,'Position',[0,0,6,4]) % Set figure width (6 in.) and height (4 in.)
cd n:\thesis\plots % Select where to save
print -dpdf plot.pdf % Save as PDF
```