DATA 1050 Cheatsheet · Samuel S. Watson

Data

1 All data stored on a computer is ultimately a sequence of bits (0/1). These bits are endowed with meaning based on a **specification**.

2 Types of data include plain text, documents, images, video, audio, tabular data, and many others.

3 Common hierarchical (nested) data formats include XML and JSON:

```
XML:
<svg version="1.1"
     baseProfile="full"
     width="300" height="200"
     xmlns="http://www.w3.org/2000/svg">
  <rect width="100%" height="100%" fill="red" />
  <circle cx="150" cy="100" r="80" fill="green" />
  <text x="150" y="125" font-size="60" fill="white">SVG</text>
</svg>

JSON:
{
   "layout":{
      "showlegend": false,
      "xaxis":{
         "range":[
            0.73,
            10.27
         ],
         "domain":[
            0.03619130941965587,
            0.9934383202099738
         ],
         "linecolor":"rgba(0, 0, 0, 1.000)",
         "tickcolor":"rgb(0, 0, 0)",
         "tickfont":{
            "color":"rgba(0, 0, 0, 1.000)",
            "size":11
         }
      }
   }
}
```

4 Tabular data formats include **CSV** and **Parquet**. CSV is a plain text format that uses commas to separate entries and newlines to separate rows. Parquet is a **binary** format (looks like gibberish if you interpret the bits of the file as plain text) which is faster and more space efficient.

Data Systems

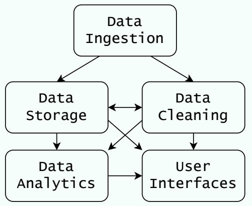
1 Organizations use a wide variety of technologies to manage their data. Organizations' concerns around data include how and where to store the data, how to access the data, how to perform calculations on the data, how to process the data and how and when to cache intermediate results, how to display the data to make it actionable, and many others.

2 **Databases** are used to store structured data, because they are designed to provide guarantees around data integrity and to provide rich access to the data.

3 **Bucket storage** in the cloud is useful for files which are large and are not structured enough to go in a database (e.g., image files, video files, PDF files).

4 A **data warehouse** is a data system in an organization which is highly structured and carefully curated. A **data lake** is a central but less structured and/or less curated repository of data collected by the organization.

5 Data ingestion, storage, cleaning, analytics, and UIs are often related in complex ways (not a simple pipeline):

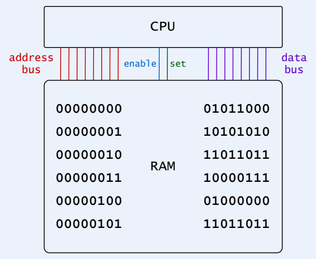


How computers work

1 Main computer components: CPU (processing), RAM (temporary storage), hard drive (persistent storage).

2 RAM consists of a sequence of 8-bit chunks called **bytes**. The index of each byte is its **address**.

3 The computer executes a program by loading its bytes into consecutive addresses in RAM and then reading the bytes in sequence. The CPU may read data stored at an address by activating the *enable* wire while putting the bits of that address on the *address bus* or write data by by using the *set* wire. The bits are read or written via the *data bus*.



4 Bytes (or chunks of bytes) may represent CPU instructions, data (like integers, floats, or characters), or RAM addresses. Some instructions can tell the CPU to jump to a different location in RAM and continue reading bytes from there.

5 CPU operations are synchronized by a **clock generator**, which fires about a billion times a second. Machine integer operations can be executed in 1-3 clock cycles, while more complex operations (like floating point division) can take more like 30 cycles.

6 When you write code in a compiled language (like C, C++, Rust, Go, Haskell, OCaml, etc.), you create an executable file to be directly executed by the computer. For programs written in Python, the executable is not the program you wrote but the *Python runtime system*. The Python runtime **interprets** your code and changes the way that it executes accordingly. Other languages that use runtimes include Julia, R, Java, C#, and Javascript.

7 Many languages (Julia, Java, C#, Javascript, et al) compile parts of your code to machine code *as the program executes*; this is called **just-in-time compilation**. Neither Python nor R is JIT-compiled unless you're using a package for that purpose (like Numba) or a non-standard interpreter (like PyPy).

8 Interpreting code is typically much slower than executing compiled code (typically 5x-30x). Python, R, and MATLAB manage reasonable performance by connecting to compiled libraries—usually written in C, C++, or Fortran—for compute-intensive tasks. This is why vectorization is an important performance technique in these languages.

The shell

1 Bytes stored on the hard drive are organized into **files**. Files are organized hierarchically into an arbitrarily nested collections of **directories** (also known as *folders*).

2 The operating system customarily handles each file according to its *file type*, which is customarily indicated by its **file extension** (like `.pdf` in `resume.pdf`).

3 You can interact programmatically with your file system using a program called a **shell**. On Unix or macOS, the shell is `bash` or a close relative.

4 Important shell commands include

- `pwd` - print the current working directory
- `cd` - change current working directory
- `ls` - list the contents of the current directory
- `tree` - show contents of current working directory (recursively)
- `cat` - print the contents of a file
- `head` - print the first so many lines or characters of a file
- `mv` - move a file
- `cp` - copy a file
- `touch` - create a file or update its last-modified time
- `curl` - make a request to a URL
- `wc` - count words/lines/characters in a file
- `grep` - search for text in file contents
- `code` - open a file in VS Code

5 Set Bash variables like `MY_FAVORITE_NUMBER=3`. You can access a variable with a dollar sign, like `echo $MY_FAVORITE_NUMBER`.

6 Add the line

```
export PATH="/Users/jovyan/anaconda3/bin:\$PATH"
```

to your `/.bash_profile` file to add `/Users/jovyan/anaconda3/bin` to your `PATH` variable (if you want to be able to execute programs in that folder by name from the command line).

7 **Piping**. The output of a command like `echo $PATH`, which prints to the screen by default, may be redirected to a file using the operators `>` or `>>` or fed as input to another bash command on the same line using the pipe operator `|`.

8 **Glob patterns**. You can perform an action on many files by including an asterisk in the file name. For example, `mv img*.png frames/` moves every file in the current directory whose name starts with `img` and ends with `png` into the 'frames' subdirectory of the current directory.

Using Python

1 **Tooling** for a programming language refers to anything we can use to make the development experience more pleasant (more efficient, more interactive, less uncertain, etc.).

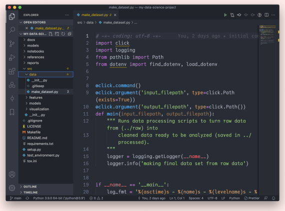
2 Jupyter is a popular development environment which provides researchers with tools for combining exposition and code into a single document called a **Jupyter notebook**. Under the hood, the file contents of a Jupyter notebooks is a JSON string.

3 Jupyter supports many **magic commands** which are not part of the Python language but which allow us to do various convenient things. For example, the `%%sql` magic causes the contents of the cell to be interpreted as SQL code.

4 Jupyter has an edit mode for entering text in cells and a command mode for manipulating cells (for example, merging or deleting cells). If there's a blinking cursor in a cell, the current mode is edit, and otherwise the current mode is command. Switching between modes is accomplished with the `escape` key (edit to command mode) and the `enter` key (command to edit mode).

5 Jupyter has many keyboard shortcuts which are worth learning. Cells are deleted in command mode with two strokes of the `d` key. You can highlight cells in command mode by holding shift and using your arrow keys, and you can merge the highlighted cells into a single cell using shift-m. Insertion of new cells is accomplished with either `a` (insert cell above) or `b` (insert cell below) in command mode. Cells can be switched between Markdown (`m`) and code (`y`) in command mode.

6 **VS Code** is a text editor with many features and extensions to support development in many languages, including Python. It has better support than Jupyter for working with multiple files, debugging (stepping through code), refactoring (changing the structure of your code), and version control.



```
# -*- coding: utf-8 -*-      You, 2 days ago • initial co
import click
import logging
from pathlib import Path
from dotenv import find_dotenv, load_dotenv


@click.command()
@click.argument('input_filepath', type=click.Path
(exists=True))
@click.argument('output_filepath', type=click.Path())
def main(input_filepath, output_filepath):
    """ Runs data processing scripts to turn raw data
    from (../raw) into
        cleaned data ready to be analyzed (saved in ../
        processed).
    """
    logger = logging.getLogger(__name__)
    logger.info('making final data set from raw data')


if __name__ == '__main__':
    log_fmt = '%(asctime)s - %(name)s - %(levelname)s - %
```

7 You can do nearly everything in VS Code through the **command palette** (`command+shift+p`). Start typing words relevant to what you want to do and select the desired option.

8 Install the Python and Jupyter extensions from the Marketplace (left sidebar), and you can execute Python code (`shift+enter`, in a `.py` file), inspect variables (in the Jupyter panel that opens when you execute code), autocomplete variable names, debug (place a red dot in the gutter and then click the bug icon in the sidebar), and run your `pytest` tests.

DATA 1050 Cheatsheet · Samuel S. Watson

Version Control with Git

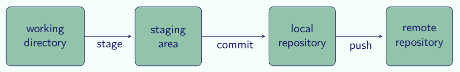
1 Git is the main software that developers use to version control their code.

2 It works using a combination of a command line program (`git`) and a folder called `.git` in the top-level directory of each project being version controlled.

3 You create a new repo by doing `git init` in the desired directory. Then create a file, stage it by doing `git add --all`, and create your initial commit with `git commit -m "initial commit"`.

4 Your version history consists of a collection of **commits** (snapshots of your project directory) which are connected via parent-child relationships.

5 Your changes go through a sequence of zones: files in your working directory are initially untracked by Git. Then you stage them with `git add` to prepare a tidy commit. Then you create a new commit in your version history with `git commit -m "commit message"`. Lastly, you update GitHub's copy of your version history with `git push`.

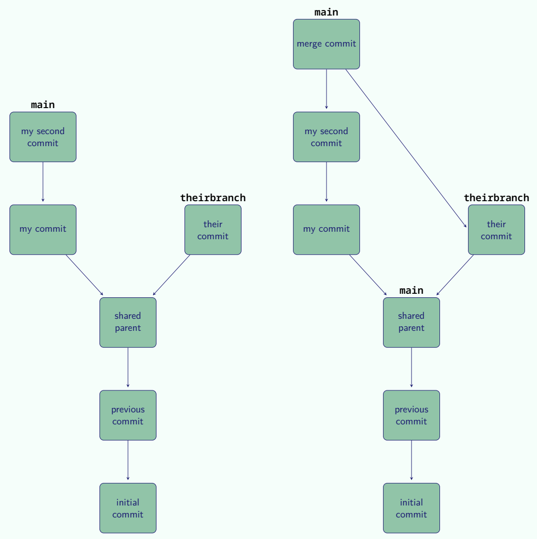


6 You will receive code to set your remote repository to a particular repo on GitHub when you create that repo on GitHub. You can see the current remote URL with `git remote -v`.

7 A **branch** is a pointer to a particular commit. You start a new line of work by creating a new branch that points to the commit you want to start from, applying the desired changes, and making new commits.

8 **Checking out** a branch sets the state of your working directory to the state of the commit that the branch points to. To preserve any unsaved work in your working directory, do a `git stash`. Put that work back into your working directory later with `git stash apply`. You will also want to stash when you `git pull` to get the latest copy of your code from GitHub.

9 You can **merge** a branch into yours to bring in that branch's changes (the ones added since the most recent common ancestor). Here's what it looks like if we merge `theirbranch` into `main`:



Relational data

1 A **relation** is a set of named tuples (with a common set of names) and can be visualized as a table with column headers. The **relational data model** represents data as a collection of **relations**.

2 **Relational algebra** is a collection of mathematical operations that may be be performed on relations:

- **Projection**. Subsetting columns.
- **Restriction**. Subsetting rows based on a condition.
- **Cartesian product**. Forming every possible concatenation of a tuple from one relation with a tuple from a second relation.
- **Sorting**. Ordering tuples according to a condition.
- **Grouping and aggregation**. Applying an aggregation function to the values in a column, potentially after grouping the tuples in the relation (partitioning them according to a condition).
- **Renaming**. Changing the name of one of the fields in a relation (changing a column header, essentially).

SQL Queries (PostgreSQL)

1 **SQL** (Structured Query Language) is the standard language for performing the relational algebra operations on tables stored in a relational database.

2 SQL is **declarative**, meaning that we express the result we want to obtain, not the steps the system is supposed to take to achieve that result.

3 SQL input consists of a sequence of **commands**. A command is composed of a sequence of **tokens** and is terminated by a semicolon.

4 A token can be a *keyword*, an *identifier*, a *literal*, or a *special character symbol*. Tokens are separated by whitespace.

5 **Keywords** are reserved words in the language with special meaning. In the statement `SELECT * FROM birds;`, both `SELECT` and `FROM` are keywords.

6 **Identifiers** specify tables, columns, or other database objects (depending on context). `birds` is an identifier which specifies which table we're selecting from.

7 Identifiers may be surrounded by double quotes to ensure they are not interpreted as keywords and to allow them to use otherwise disallowed characters (like whitespace).

8 String literals in SQL are enclosed in single quotes. Numeric literals can be entered like `4`, `3.2`, or `1.925e-3`.

9 Queries use the `SELECT` keyword. The basic structure of a `SELECT` statement is

```
SELECT [select_list] FROM [table_expression] [sort_specification];
```

The table expression is evaluated and then passed to the select list. The sort specification (if present) then processes the resulting rows before they are returned.

10 The **table expression** is an expression that returns a table, like a table name or another `SELECT` statement enclosed in parentheses.

11 The **select list** is a comma-separated list of *value expressions*, which may consist of column identifiers, constant literals, or expressions involving function calls and operators. In this context, the asterisk is a special character meaning "all columns".

12 Each value expression may be assigned a specific name using the `AS` keyword.

```
SELECT
  common_name,
  LENGTH(common_name) AS name_length
  victory_points + egg_capacity AS total_points,
FROM
  birds;
```

common_name	victory_points	egg_capacity
American Robin	1	4
Cedar Waxwing	3	3
Ash-Throated Flycatcher	4	4
Southern Cassowary	4	4
Common Nightingale	3	4

↓

common_name	name_length	total_points
American Robin	14	5
Cedar Waxwing	13	6
Ash-Throated Flycatcher	23	8
Southern Cassowary	18	8
Common Nightingale	18	7

13 The table expression may be modified by further clauses indicated by keywords like `WHERE` or `GROUP BY` or `HAVING`.

14 The sort specification is a clause of the form `ORDER BY [value_expression] [ASC|DESC]`, where the value indicated by the value expression is evaluated for each row and used to perform the sort:

```
SELECT
    *
FROM
    birds
WHERE
    "set" = 'core' AND wingspan > 25
ORDER BY
    wingspan DESC;
```

common_name	set	wingspan
American Robin	core	43
Cedar Waxwing	core	25
Ash-Throated Flycatcher	core	30
Southern Cassowary	oceania	NULL
Common Nightingale	european	23

↓

common_name	set	wingspan
American Robin	core	43
Ash-Throated Flycatcher	core	30

15 *Grouping* by a value expression partitions the tuples in a relation into groups of equal value. If the table expression in a `SELECT` statement has been grouped, then each entry in the select list must be either a value that was grouped on or a call to an **aggregate** function (like `SUM`, `AVG`, `MAX`, `MIN`, or `COUNT`, which reduces a column of values to a single value).

```
SELECT fruit,
       MAX(LENGTH(common_name)) AS max_name_length
FROM birds
GROUP BY fruit;
```

common_name	fruit
American Robin	1
Cedar Waxwing	2
Ash-Throated Flycatcher	1
Southern Cassowary	2
Common Nightingale	1

↓

common_name	fruit
American Robin	1
Ash-Throated Flycatcher	1
Common Nightingale	1
Cedar Waxwing	2
Southern Cassowary	2

↓

fruit	max_name_length
1	23
2	18

16 Filter results from a grouped and aggregated relation using a `HAVING` clause.

17 Use `LIMIT [limit] OFFSET [offset]` after an `ORDER BY` clause to return at most `limit` records beginning at index `offset`.

18 Name a temporary table using `WITH`. Example: select every card from whichever expansion set has the largest average egg capacity:

```
WITH set_eggs AS (
    SELECT "set",
           AVG(egg_capacity) AS avg_eggs
    FROM birds
    GROUP BY "set"
    ORDER BY avg_eggs DESC LIMIT 1
)
SELECT * FROM birds
WHERE "set" IN (SELECT "set" FROM set_eggs);
```

19 A comma-separated list of two relations denotes their **Cartesian product**. To look at every (bird card, bonus card) combination:

birds		
`common_name`	`set`	`wingspan`
American Robin	core	43
Cedar Waxwing	core	25
Ash-Throated Flycatcher	core	30
Southern Cassowary	oceania	NULL
Common Nightingale	european	23
Sulphur-Crested Cockatoo	oceania	103

bonus_cards	
`name`	`condition`
Passerine Specialist	wingspan ≤ 30
Large Bird Specialist	wingspan > 64

$\downarrow$

`SELECT * FROM birds, bonus_cards;`				
`common_name`	`set`	`wingspan`	`name`	`condition`
American Robin	core	43	Passerine Specialist	wingspan ≤ 30
Cedar Waxwing	core	25	Passerine Specialist	wingspan ≤ 30
Ash-Throated Flycatcher	core	30	Passerine Specialist	wingspan ≤ 30
Southern Cassowary	oceania	NULL	Passerine Specialist	wingspan ≤ 30
Common Nightingale	european	23	Passerine Specialist	wingspan ≤ 30
Sulphur-Crested Cockatoo	oceania	103	Passerine Specialist	wingspan ≤ 30
American Robin	core	43	Large Bird Specialist	wingspan > 65
Cedar Waxwing	core	25	Large Bird Specialist	wingspan > 65
Ash-Throated Flycatcher	core	30	Large Bird Specialist	wingspan > 65
Southern Cassowary	oceania	NULL	Large Bird Specialist	wingspan > 65
Common Nightingale	european	23	Large Bird Specialist	wingspan > 65
Sulphur-Crested Cockatoo	oceania	103	Large Bird Specialist	wingspan > 65

20 Cartesian products are usually combined with a `WHERE` clause. To find which (bird, bonus card) combinations actually yield bonuses:

```
SELECT * FROM birds, bonus_cards
WHERE wingspan <= 30 AND condition = 'wingspan ≤ 30'
   OR wingspan > 65 AND condition = 'wingspan > 65';
```

`common_name`	`set`	`wingspan`	`name`	`condition`
Cedar Waxwing	core	25	Passerine Specialist	wingspan ≤ 30
Ash-Throated Flycatcher	core	30	Passerine Specialist	wingspan ≤ 30
Common Nightingale	european	23	Passerine Specialist	wingspan ≤ 30
Sulphur-Crested Cockatoo	oceania	103	Large Bird Specialist	wingspan > 65

21 Cartesian products with restrictions are important enough to warrant their own syntax: `[table1] JOIN [table2] ON [condition]`

```
SELECT * FROM birds JOIN bonus_cards
ON wingspan <= 30 AND condition = 'wingspan ≤ 30'
   OR wingspan > 65 AND condition = 'wingspan > 65';
```

22 Joins come in several flavors:

- `JOIN` or `INNER JOIN`. Cartesian product followed by restriction.
- `LEFT OUTER JOIN`. Inner join followed by adding a single row for each row from the first table completely eliminated by the restriction. Those rows get `NULL` values for second-table fields.

`SELECT * FROM birds LEFT OUTER JOIN bonus_cards;`				
`common_name`	`set`	`wingspan`	`name`	`condition`
Cedar Waxwing	core	25	Passerine Specialist	wingspan ≤ 30
Ash-Throated Flycatcher	core	30	Passerine Specialist	wingspan ≤ 30
Common Nightingale	european	23	Passerine Specialist	wingspan ≤ 30
Sulphur-Crested Cockatoo	oceania	103	Large Bird Specialist	wingspan > 65
American Robin	core	43	NULL	NULL
Southern Cassowary	oceania	NULL	NULL	NULL

- `RIGHT OUTER JOIN`. Same but for eliminated rows from the *second* table.
- `FULL OUTER JOIN`. Same but for eliminated rows from *either* table.
- `CROSS JOIN`. Cartesian product with no restriction.
- `NATURAL JOIN`. Inner join on equality comparison of all pairs of identically named fields.

23 We can take a union of tuples in two relations (with the same field names) using the `UNION` operator. We can take a set difference using `EXCEPT` and the intersection using `INTERSECT`.

24 The syntax for a table literal is `VALUES (row1), (row2), (row3);` To add two rows manually:

```
(SELECT common_name, "set" FROM birds)
UNION
(VALUES ('Western Tanager', 'core'),
        ('Scissor-Tailed Flycatcher', 'core'));
```

SQL: Modifying Data

1 To add rows to a database:

```
INSERT INTO
    birds(common_name, "set")
VALUES
    ('Western Tanager', 'core'),
    ('Scissor-Tailed Flycatcher', 'core');
```

2 To update rows to a database:

```
UPDATE
    birds
SET
    wingspan = 0
WHERE
    wingspan IS NULL;
```

3 To delete rows:

```
DELETE FROM
    birds
WHERE
    "set" NOT IN ('core', 'oceania', 'european');
```

SQL: Managing Tables

1 Creating a new table. To make a new table called `birds` a text field `common_name` which will be used as a primary key, a text field `set` which is a foreign key for the `name` column in another table called `expansions`, and an integer field `wingspan` which should not be allowed to be negative:

```
CREATE TABLE birds (
    common_name TEXT PRIMARY KEY,
    "set" TEXT REFERENCES expansions(name),
    wingspan INTEGER CHECK (wingspan >= 0),
);
```

2 PostgreSQL

- `BIGINT/INT8` signed eight-byte integer
- `INTEGER/INT/INT4` signed four-byte integer
- `DOUBLE PRECISION/FLOAT8` double precision floating-point number (8 bytes)
- `REAL/FLOAT4` single precision floating-point number (4 bytes)
- `BOOLEAN/BOOL` logical Boolean (true/false)
- `VARCHAR(n)` variable-length character string (max `n` characters)
- `TEXT` variable-length character string
- `DATE` calendar date (year, month, day)
- `MONEY` currency amount
- `NUMERIC [ (p, s) ]` exact numeric of selectable precision
- `TIMESTAMP` date and time
- `UUID` universally unique identifier

3 To drop a table: `DROP TABLE [table_name];`

4 To remove all data from a table: `TRUNCATE TABLE [table_name];`

5 To add a column: `ALTER TABLE [table_name] ADD [column_name column_type];`

SQL: Setup

1 Easiest way to create a free cloud Postgres instance: Go to `supabase.io` > Log in with GitHub > Create an Organization > Create a New Project > [wait a few minutes, and in the meantime add the line `export DATABASE_PWD="your-pwd-here"` to your bash profile] > Go into the new project > Settings (gear icon) > Database > Connection String (bottom) > PSQL > Copy.

2 macOS local installation: `https://postgresapp.com/`. Instructions on the landing page for finding your connection string. To install locally on Windows: `https://www.postgresql.org/download/windows/`.

3 To connect from a Python session, paste the connection string replacing `[YOUR-PASSWORD]` with `{pwd}`, like this:

```
import sqlalchemy
import os
pwd = os.envget("DATABASE_PWD") # retrieve password from bashrc
connection_string = (
   f"postgresql://postgres:{pwd}@"
    "db.bijsjfasiwdlfkjasdfot.supabase.co:5432/postgres"
) # should be your connection string instead
engine = create_engine(connection_string)
connection = engine.connect()
sql = "SELECT * FROM pg_catalog.pg_tables LIMIT 10;"
connection.execute(sql).fetchall()
```

4 Create a new table in the database from a Pandas dataframe:

```
import pandas as pd
df = pd.read_csv("https://bit.ly/iris-dataset")
df.to_sql("iris", con=engine)
```