CERTIFICATE

It has been certified that Lavanya Trivedi, who is enrolled in XIIth Sci α during the academic year 2024-2025, has finished her physics investigatory project on the topic "Hangman Game – Periodic Table edition using Pandas and CSV" and has provided an unambiguous presentation of the results found in her project report.

Mrs. Anuradha Patri (IP)

_____

Signature

External examiner

_____

Signature

Dr. (Mrs.) Shruti Gupta

(Principal)

_____

Signature

ACKNOWLEDGEMENT

My sincere efforts have made me to accomplish the task of completing the investigatory project on the "Hangman Game – Periodic Table edition using Pandas and CSV". However, it would not have been possible without the kind support and help of many individuals.

I extend my sincere thank to Dr. (Mrs.) Shruti Gupta, our director principal ma'am for her consistent encouragement.

I would like to thank our HOD of physics, Mrs. Anuradha Patri ma'am for her never-ending support, unwavering guidance and supervision during the completion of this project.

I would like to thank our lab assistant who for helping me in this project.

I am also thankful to parents for their support morally and academically.

Lavanya Trivedi

XIIth Sci α

DECLARATION

I hereby declare that the project work entitled "Hangman Game – Periodic Table edition using Pandas and CSV" that I turned in to **Brilliant Public School**, Bilaspur is a record of an investigative project carried out with the help of Mrs. Anuradha Patri, the school's IP department teacher and **lab assistant who**, the lab assistant. The content and pictures provided are the outcome of my work.

Lavanya Trivedi

Class 12th Sci α

Pandas (styled as pandas) are a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.

It is free software released under the three-clause BSD license.

The name is derived from the term "panel data", an econometrics term for data sets that include observations over multiple time periods for the same individuals, as well as a play on the phrase "Python data analysis".

The development of Pandas introduced into Python many comparable features of working with Data Frames that were established in the R programming language. The library is built upon another library, NumPy.

History

Developer Wes McKinney started working on Pandas in 2008 while at AQR Capital Management out of the need for a high performance, flexible tool to perform quantitative analysis on financial data. Before leaving AQR he was able to convince management to allow him to open source the library.

Another AQR employee, Chang She, joined the effort in 2012 as the second major contributor to the library.

In 2015, Pandas signed on as a fiscally sponsored project of NumFOCUS, a 501(c)(3) nonprofit charity in the United States.

Data Model

Pandas is built around data structures called *Series* and *DataFrames*. Data for these collections can be imported from various file formats such as comma-separated values (CSV), JSON, Parquet, SQL database tables or queries, and Microsoft Excel.

What is Series?

A *Series* is a 1-dimensional data structure built on top of NumPy's array. Unlike in NumPy, each data point has an associated label. The collection of these labels is called an index. Series can be used arithmetically, as in the statement

```
series_3 = series_1 + series_2
```

This will align data points with corresponding index values in series_1 and series_2, and then add them together to produce new values in series_3.

What is DataFrame?

A *DataFrame* is a 2-dimensional data structure of rows and columns, similar to a spreadsheet, and analogous to a Python dictionary mapping column names (keys) to Series (values), with each Series sharing an index.

DataFrames can be concatenated together or "merged" on columns or indices in a manner similar to joins in SQL.

Pandas implement a subset of relational algebra, and support one-to-one, many-to-one, and many-to-many join. Pandas also support the less common *Panel* and *Panel4D*, which are 3-dimensional and 4-dimension data structures respectively.

Users can transform or summarize data by applying arbitrary functions. Since Pandas is built on top of NumPy, all NumPy functions work on Series and DataFrames as well. Pandas also include built-in operations for arithmetic, string manipulation, and summary statistics such as mean, median, and standard deviation. These built-in functions are designed to handle missing data, usually represented by the floating-point value NaN.

Subsets of data can be selected by column name, index, or Boolean expressions. For example,

```
df[df['col1'] > 5]
```

This will return all rows in the DataFrame (say df) for which the value of the column col1 exceeds 5. Data can be grouped together by a column value, as in

```
df['col1'].groupby(df['col2']).
```

Pandas includes support for time series, such as the ability to interpolate values and filter using a range of timestamps (e.g. `data['1/1/2023':'2/2/2023']`, will return all dates between January 1st and February 2nd). Pandas represents missing time series data using a special *NaT* (Not a Timestamp) object, instead of the NaN value it uses elsewhere.

Indices

By default, a Pandas index is a series of integers ascending from 0, similar to the indices of Python arrays. However, indices can use any NumPy data type, including floating point, timestamps, or strings.

Syntax of Pandas for mapping index values to relevant data is the same syntax Python uses to map dictionary keys to values.

For example, if s is a Series, s['a'] will return the data point at index a. Unlike dictionary keys, index values are not guaranteed to be unique. If a Series uses the index value a for multiple data points, then s['a'] will instead return a new Series containing all matching values.

A DataFrame's column names are stored and implemented identically to an index. As such, a DataFrame can be thought of as having two indices: one column-based and one row-based. Because column names are stored as an index, there are also not required to be unique.

If data is a Series, then data['a'] returns all values with the index value of a. However, if data is a DataFrame, then data['a'] returns all values in the column(s) named a. To avoid this ambiguity, Pandas supports the syntax data.loc['a'] as an alternative way to filter using the index.

Pandas also support the syntax data.loc[n], which always takes an integer *n* and returns the *nth* value, counting from 0. This allows a user to act as though the index is an array-like sequence of integers, regardless of how it's actually defined.

Pandas support hierarchical indices with multiple values per data point. An index with this structure, called a MultiIndex, allows a single DataFrame to represent multiple dimensions, similar to a pivot table in Microsoft Excel. Each level of a MultiIndex can be given a unique name. In practice, data with more than 2 dimensions is often represented using DataFrames with hierarchical indices, instead of the higher-dimension *Panel* and *Panel4D* data structures.

Comma – Separated Values

Comma-separated values (CSV) is a text file format that uses commas to separate values, and newlines to separate records. A CSV file stores tabular data (numbers and text) in plain text, where each line of the file typically represents one data record. Each record consists of the same number of fields, and these are separated by commas in the CSV file. If the field delimiter itself may appear within a field, fields can be surrounded with quotation marks.

The CSV file format is one type of delimiter-separated file format. Delimiters frequently used include the comma, tab, space, and semicolon. Delimiter-separated files are often given a ".csv" extension even when the field separator is not a comma. Many applications or libraries that consume or produce CSV files have options to specify an alternative delimiters.

The lack of adherence to the CSV standard RFC 4180 necessitates the support for a variety of CSV formats in data input software. Despite this drawback, CSV remains widespread in data applications and is widely supported by a variety of software, including common spreadsheet applications such as Microsoft Excel. Benefits cited in favor of CSV include human readability and the simplicity of the format.

Applications

CSV is a common data exchange format that is widely supported by consumer, business, and scientific applications. Among its most common uses is moving tabular data between programs that natively operate on incompatible (often proprietary or undocumented) formats.

For example, a user may need to transfer information from a database program that stores data in a proprietary format, to a spreadsheet that uses a completely different format. Most database programs can

export data as CSV. Most spreadsheet programs can read CSV data, allowing CSV to be used as an intermediate format when transferring data from a database to a spreadsheet.

CSV is also used for storing data. Common data science tools such as Pandas include the option to export data to CSV for long-term storage. Benefits of CSV for data storage include the simplicity of CSV makes parsing and creating CSV files easy to implement and fast compared to other data formats, human readability making editing or fixing data simpler, and high compressibility leading to smaller data files.

Alternatively, CSV does not support more complex data relations and makes no distinction between null and empty values, and in applications where these features are needed other formats are preferred.

History

Comma-separated values is a data format that predates personal computers by more than a decade: the IBM Fortran (level H extended) compiler under OS/360 supported CSV in 1972.

List-directed ("free form") input/output was defined in FORTRAN 77, approved in 1978. List-directed input used commas or spaces for delimiters, so unquoted character strings could not contain commas or spaces.

The term "comma-separated value" and the "CSV" abbreviation were in use by 1983. The manual for the Osborne Executive computer, which bundled the SuperCalc spreadsheet, documents the CSV quoting convention that allows strings to contain embedded commas, but the manual does not specify a convention for embedding quotation marks within quoted strings.

Comma-separated value lists are easier to type (for example into punched cards) than fixed-column-aligned data, and they were less prone to producing incorrect results if a value was punched one column off from its intended location.

Comma separated files are used for the interchange of database information between machines of two different architectures. The plain-text character of CSV files largely avoids incompatibilities such as byte-

order and word size. The files are largely human-readable, so it is easier to deal with them in the absence of perfect documentation or communication.

The main standardization initiative—transforming "*de facto* fuzzy definition" into a more precise and *de jure* one—was in 2005, with RFC 4180, defining CSV as a MIME Content Type.

Later, in 2013, some of RFC 4180's deficiencies were tackled by a W3C recommendation.

In 2014 IETF published RFC 7111 describing the application of URI fragments to CSV documents. RFC 7111 specifies how row, column, and cell ranges can be selected from a CSV document using position indexes.

In 2015 W3C, in an attempt to enhance CSV with formal semantics, publicized the first *drafts of recommendations* for CSV metadata standards, which began as *recommendations* in December of the same year.

General functionality

The greatest use cases for CSV formats are collections or sequences of records where every record has the same set of fields. This is equivalent to one relation in a relational database or to data in a standard spreadsheet, but without any calculations.
The format, which is frequently used to transfer data across computers with various internal word sizes, data formatting requirements, and other factors, dates back to the early days of business computing. CSV files are hence ubiquitous across all computer platforms.
A comma is used in CSV files, which are delimited text files, while alternative separators can be used in some CSV import/export tool implementations.

For example, the use of a "Sep=^" row as the first row in the *.csv file will cause Excel to open the file expecting caret "^" to be the separator instead of comma ",").

Simple CSV implementations may prohibit field values that contain a comma or other special characters such as newlines. More sophisticated CSV implementations permit them, often by requiring “” (double quote) characters around values that contain reserved characters (such as commas, double quotes, or less commonly, newlines). Embedded double quote characters may then be represented by a pair of consecutive double quotes, or by prefixing a double quote with an escape character such as a backslash (for example in Sybase Central).

CSV formats are not limited to a particular character set. They work just as well with Unicode character sets (such as UTF-8 or UTF-16) as with ASCII (although particular programs that support CSV may have their own limitations). CSV files normally will even survive naïve translation from one character set to another (unlike nearly all proprietary data formats). CSV does not, however, provide any way to indicate what character set is in use, so that must be communicated separately, or determined at the receiving end (if possible).

Databases that include multiple relations cannot be exported as a single CSV file. Similarly, CSV cannot naturally represent hierarchical or object-oriented data. This is because every CSV record is expected to have the same structure. CSV is therefore rarely appropriate for documents created with HTML, XML, or other markup or word-processing technologies.

Statistical databases in various fields often have a generally relation-like structure, but with some repeatable groups of fields. For example, health databases such as the Demographic and Health Survey typically repeat some questions for each child of a given parent (perhaps up to a fixed maximum number of children). Statistical analysis systems often include utilities that can "rotate" such data.

For example, a "parent" record that includes information about five children can be split into five separate records, each containing (a) the information on one child, and (b) a copy of all the non-child-specific information. CSV can represent either the "vertical" or "horizontal" form of such data.

In a relational database, similar issues are readily handled by creating a separate relation for each such group, and connecting "child" records to the related "parent" records using a foreign key (such as an ID number or name for the parent). In markup languages such as XML, such groups are typically enclosed within a parent element and repeated as necessary (for example, multiple <child> nodes within a single <parent> node). With CSV there is no widely accepted single-file solution.

Python

Guido Van Rossum

Python is a high-level, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation.

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly procedural), object-oriented and functional programming. It is often described as a "batteries included" language due to its comprehensive standard library.

Guido van Rossum began working on Python in the late 1980s as a successor to the ABC programming language and first released it in 1991 as Python 0.9.0. He is also the original creator of IDLE platform.

Python 2.0 was released in 2000. Python 3.0, released in 2008, was a major revision not completely backward-compatible with earlier versions. Python 2.7.18, released in 2020, was the last release of Python 2.

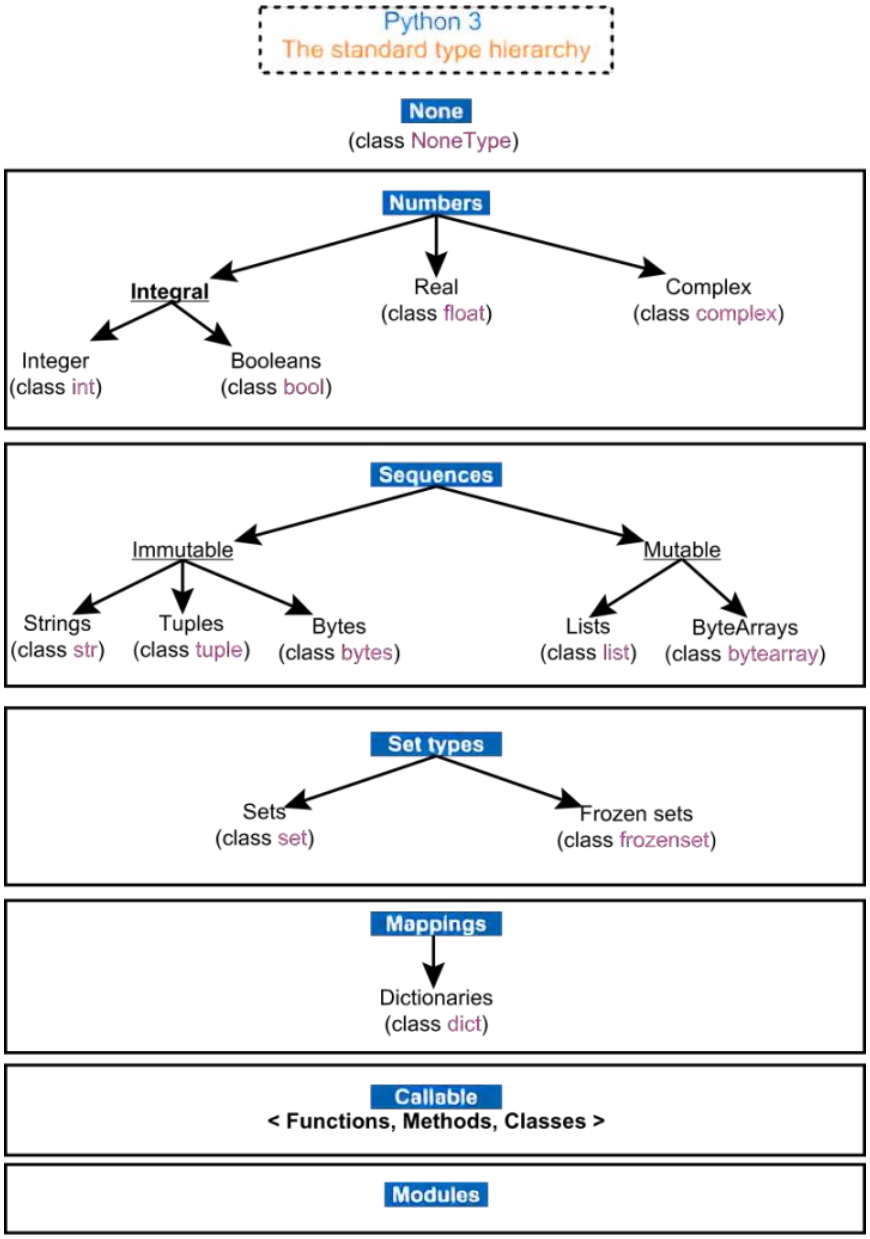
Python consistently ranks as one of the most popular programming languages, and has gained widespread use in the machine learning community.

History

Python, invented in the late 1980s by Guido van Rossum, is a successor to the ABC programming language. It was implemented in 1989 and has since undergone several releases, including Python 2.0, Python 3.0, and Python 3.13. Python 2.7 has reached end-of-life in 2015, and only 3.8 and later are supported. As of October 2023, Python 3.12 is the stable release, with significant changes including increased program execution speed and improved error reporting. Python 3.8 is the oldest supported version, with Python 3.13 introducing an incremental garbage collector and an experimental JIT compiler.

Design philosophy and features

Python is a multi-paradigm programming language that supports object-oriented, structured, functional, aspect-oriented, and many other paradigms. It uses dynamic typing, reference counting, and a cycle-detecting garbage collector for memory management. Python's core philosophy is summarized in the Zen of Python, which emphasizes beauty, explicitness, simplicity, complexity, and readability. Despite criticisms, Python is designed to be highly extensible via modules, making it popular for adding programmable interfaces to existing applications. Its developers aim for simplicity, less cluttered syntax, and a fun user experience.



Summary of Python 3's built-in types		
Type	Mutability	Syntax examples
bool	immutable	`True` `False`

bytearray	mutable	`bytearray(b'Some ASCII')`
		`bytearray(b"Some ASCII")`
		`bytearray([119, 105, 107, 105])`

bytes	immutable	`b'Some ASCII'`
		`b"Some ASCII"`
		`bytes([119, 105, 107, 105])`

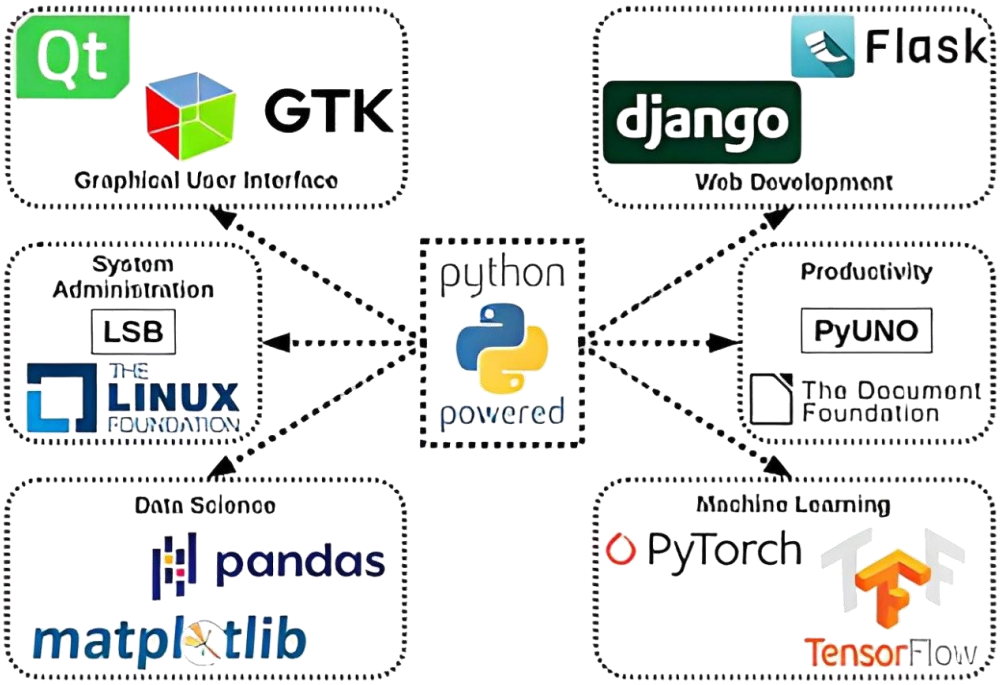
complex	immutable	`3+2.7j` `3 + 2.7j`
dict	mutable	`{'key1': 1.0, 3: False}` `{}`
types.EllipsisType	immutable	`...` `Ellipsis`
float	immutable	`1.33333`
frozenset	immutable	`frozenset([4.0, 'string', True])`
int	immutable	`42`
list	mutable	`[4.0, 'string', True]` `[]`
types.NoneType	immutable	`None`
types.NotImplementedType	immutable	`NotImplemented`

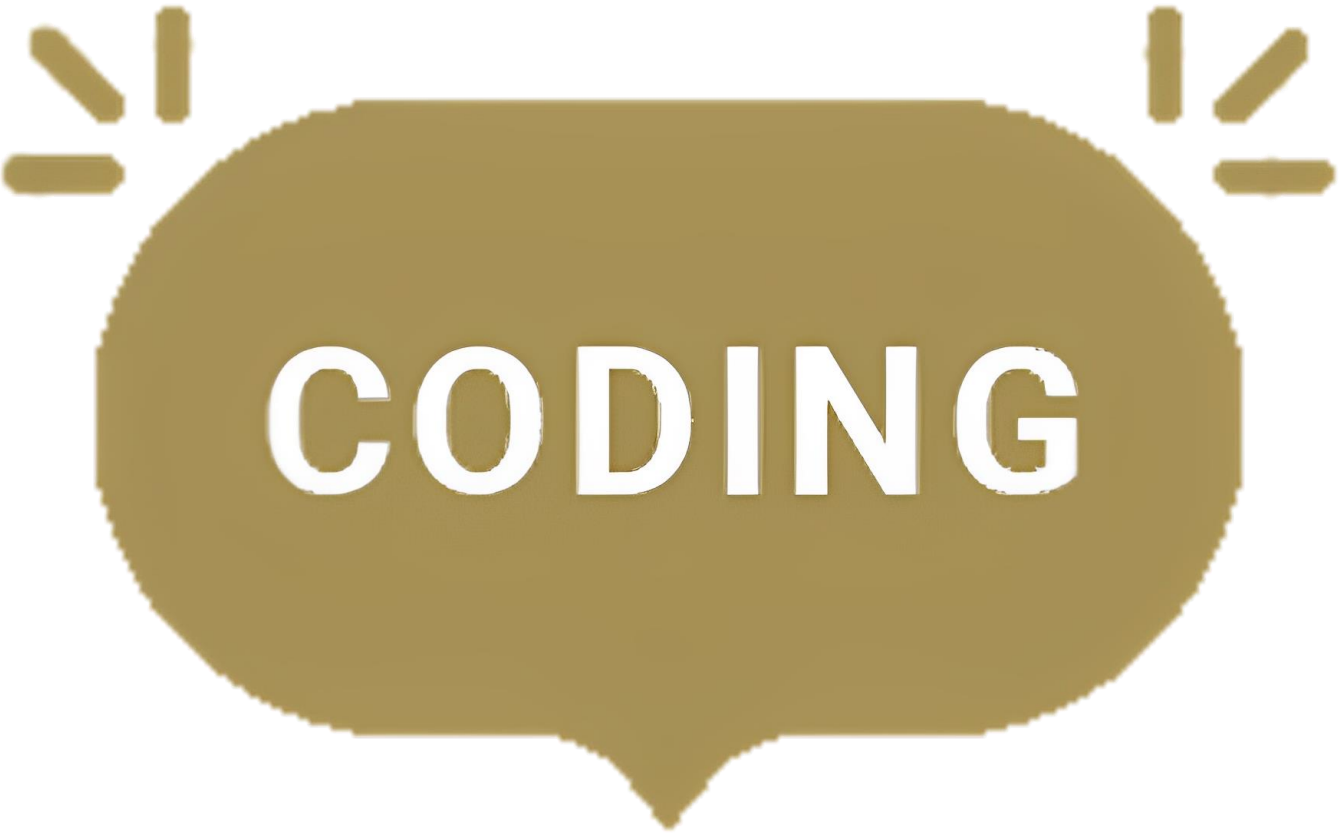
`range`	immutable	`range(-1, 10)` `range(10, -5, -2)`
`set`	mutable	`{4.0, 'string', True}` `set()`

`str`	immutable	`'Wikipedia'`
		`"Wikipedia"`
		
		`"""Spanning multiple lines"""`
		`Spanning multiple lines`

`tuple`	immutable	`(4.0, 'string', True)`
		`('single element',)`
		`()`

Uses:




CODING

Code for adding csv file:

```
import pandas as pd
# Define the data for the periodic table
periodic_table_data = {
    'Symbol': ['H', 'He', 'Li', 'Be', 'B', 'C', 'N', 'O', 'F', 'Ne',
               'Na', 'Mg', 'Al', 'Si', 'P', 'S', 'Cl', 'Ar',
               'K', 'Ca', 'Sc', 'Ti', 'V', 'Cr', 'Mn', 'Fe', 'Co', 'Ni',
               'Cu', 'Zn', 'Ga', 'Ge', 'As', 'Se', 'Br', 'Kr', 'Rb', 'Sr',
               'Y', 'Zr', 'Nb', 'Mo', 'Tc', 'Ru', 'Rh', 'Pd', 'Ag', 'Cd',
               'In', 'Sn', 'Sb', 'Te', 'I', 'Xe', 'Cs', 'Ba', 'La', 'Ce',
               'Pr', 'Nd', 'Pm', 'Sm', 'Eu', 'Gd', 'Tb', 'Dy', 'Ho', 'Er',
               'Tm', 'Yb', 'Lu', 'Hf', 'Ta', 'W', 'Re', 'Os', 'Ir', 'Pt',
               'Au', 'Hg', 'Tl', 'Pb', 'Bi', 'Po', 'At', 'Rn', 'Fr', 'Ra',
               'Ac', 'Th', 'Pa', 'U', 'Np', 'Pu', 'Am', 'Cm', 'Bk', 'Cf',
               'Es', 'Fm', 'Md', 'No', 'Lr', 'Rf', 'Db', 'Sg', 'Bh', 'Hs',
               'Mt', 'Ds', 'Rg', 'Cn', 'Nh', 'Fl', 'Mc', 'Lv', 'Ts', 'Og'],

    'Name': ['Hydrogen', 'Helium', 'Lithium', 'Beryllium', 'Boron', 'Carbon',
             'Nitrogen', 'Oxygen', 'Fluorine', 'Neon', 'Sodium', 'Magnesium',
             'Aluminum', 'Silicon', 'Phosphorus', 'Sulfur', 'Chlorine',
             'Argon', 'Potassium', 'Calcium', 'Scandium', 'Titanium',
             'Vanadium', 'Chromium', 'Manganese', 'Iron', 'Cobalt', 'Nickel',
             'Copper', 'Zinc', 'Gallium', 'Germanium', 'Arsenic', 'Selenium',
             'Bromine', 'Krypton', 'Rubidium', 'Strontium', 'Yttrium',
             'Zirconium', 'Niobium', 'Molybdenum', 'Technetium', 'Ruthenium',
             'Rhodium', 'Palladium', 'Silver', 'Cadmium', 'Indium', 'Tin',
             'Antimony', 'Tellurium', 'Iodine', 'Xenon', 'Cesium', 'Barium',
             'Lanthanum', 'Cerium', 'Praseodymium', 'Neodymium', 'Promethium',
             'Samarium', 'Europium', 'Gadolinium', 'Terbium', 'Dysprosium',
             'Holmium', 'Erbium', 'Thulium', 'Ytterbium', 'Lutetium',
             'Hafnium', 'Tantalum', 'Tungsten', 'Rhenium', 'Osmium',
             'Iridium', 'Platinum', 'Gold', 'Mercury', 'Thallium', 'Lead',
             'Bismuth', 'Polonium', 'Astatine', 'Radon', 'Francium', 'Radium',
             'Actinium', 'Thorium', 'Protactinium', 'Uranium', 'Neptunium',
             'Plutonium', 'Americium', 'Curium', 'Berkelium', 'Californium',
             'Einsteinium', 'Fermium', 'Mendelevium', 'Nobelium', 'Lawrencium',
             'Rutherfordium', 'Dubnium', 'Seaborgium', 'Bohrium', 'Hassium',
             'Meitnerium', 'Darmstadtium', 'Roentgenium', 'Copernicium',
             'Nihonium', 'Flerovium', 'Moscovium', 'Livermorium',
             'Tennessine', 'Oganesson'],

  'AtomicMass': [1.008, 4.002602, 6.94, 9.0121831, 10.81, 12.011, 14.007,
                 15.999, 18.998403163, 20.1797, 22.98976928, 24.305,
                 26.9815385, 28.085, 30.973761998, 32.06, 35.45, 39.948,
                 39.0983, 40.078, 44.955908, 47.867, 50.9415, 51.9961,
                 54.938044, 55.845, 58.933194, 58.6934, 63.546, 65.38,
                 69.723, 72.63, 74.921595, 78.971, 79.904, 83.798, 85.4678,
                 87.62, 88.90584, 91.224, 92.90637, 95.95, 98, 101.07,
                 102.90550, 106.42, 107.8682, 112.414, 114.818, 118.710,
                 121.760, 127.60, 126.90447, 131.293, 132.90545196, 137.327,
                 138.90547, 140.116, 140.90766, 144.242, 145, 150.36, 151.964,
                 157.25, 158.92535, 162.500, 164.93033, 167.259, 168.93422,
                 173.045, 174.9668, 178.49, 180.94788, 183.84, 186.207,
                 190.23, 192.217, 195.084, 196.966569, 200.592, 204.38,
                 207.2, 208.98040, 209, 210, 222, 223, 226, 227, 232.0377,
                 231.03588, 238.02891, 237, 244, 243, 247, 247, 251, 252,
                 257, 258, 259, 262, 267, 268, 271, 270, 277, 276, 281, 280,
                 285, 284, 289, 288, 293, 294, 294],
```

```
'Nature': ['gas', 'synthetic', 'solid', 'solid','solid','solid',
           'synthetic', 'synthetic', 'gas', 'synthetic',
           'solid', 'solid', 'solid', 'solid', 'solid', 'solid', 'gas',
           'synthetic', 'solid','solid','solid','solid','solid','solid',
           'solid','solid','solid','solid','solid','solid','liquid','solid',
           'solid','solid','liquid','synthetic','solid','solid','solid',
           'solid','solid','solid','gas and radioactive','solid','solid',
           'solid','solid','solid','solid','solid','solid','solid','solid',
           'synthetic', 'liquid', 'solid','solid','solid','solid','solid',
           'gas and radioactive', 'solid and radioactive','solid','solid',
           'solid','solid','solid','solid','solid','solid','solid','solid',
           'solid','solid','solid','solid','solid','solid','solid','liquid',
           'solid','solid','solid','solid and radioactive',
           'solid and radioactive','synthetic and radioactive',
           'liquid and radioactive','solid and radioactive',
           'solid and radioactive','solid and radioactive',
           'solid and radioactive','solid and radioactive',
           'gas and radioactive','gas and radioactive','gas and radioactive',
           'gas and radioactive','gas and radioactive','gas and radioactive',
           'gas and radioactive','gas and radioactive','gas and radioactive',
           'gas and radioactive','gas and radioactive',
           'synthetic and radioactive', 'synthetic and radioactive',
           'synthetic and radioactive', 'synthetic and radioactive',
           'synthetic and radioactive', 'synthetic and radioactive',
           'synthetic and radioactive','synthetic and radioactive',
           'synthetic and radioactive', 'synthetic and radioactive',
           'synthetic and radioactive', 'synthetic and radioactive',
           'synthetic and radioactive', 'synthetic and radioactive',
           'synthetic and radioactive'
        ],

    'AtomicNumber': [1,2,3,4,5,6,7,8,9,10,
                     11,12,13,14,15,16,17,18,19,20,
                     21,22,23,24,25,26,27,28,29,30,
                     31,32,33,34,35,36,37,38,39,40,
                     41,42,43,44,45,46,47,48,49,50,
                     51,52,53,54,55,56,57,58,59,60,
                     61,62,63,64,65,66,67,68,69,70,
                     71,72,73,74,75,76,77,78,79,80,
                     81,82,83,84,85,86,87,88,89,90,
                     91,92,93,94,95,96,97,98,99,100,
                     101,102,103,104,105,106,107,108,109,110,
                     111,112,113,114,115,116,117,118]

                }

# Create a DataFrame from the data
periodic_table_df = pd.DataFrame(periodic_table_data)

# Save the DataFrame to a CSV file
periodic_table_df.to_csv("C:\\Users\\HP\\OneDrive\\Desktop\\periodic_table.csv", index=False)
```

Code for making the game (Hangman game - periodic table edition):

```
import pandas as pd
import random
import time
import pyttsx3
import matplotlib.pyplot as plt
def load_periodic_table():
    #Insert link of the location in your device where the csv file is saved.
    df = pd.read_csv("C:\\Users\\HP\\OneDrive\\Desktop\\School
Work\\Projects\\IP Project 12th\\IP-Project-12th-Hangman-game-
main\\periodic_table.csv")
    return df

def select_random_element(df, category=None, difficulty='medium'):
    if category:
        df = df[df['AtomicNumber'] == category]
    if difficulty == 'easy':
        df = df[df['Name'].str.len() <= 5]
    elif difficulty == 'medium':
        df = df[(df['Name'].str.len() > 5) & (df['Name'].str.len() <= 8)]
    elif difficulty == 'hard':
        df = df[df['Name'].str.len() > 8]
    return df.sample()

def display_word(word):
    return '_ ' * len(word)

def update_displayed_word(word, guessed_letter, displayed_word):
    new_displayed_word = ''
    for i in range(len(word)):
        if word[i].lower() == guessed_letter.lower():
            new_displayed_word += guessed_letter + ' '
        else:
            new_displayed_word += displayed_word[i*2] + ' '
    return new_displayed_word

def get_hint(element_row):
    atomic_number = element_row['AtomicNumber'].values[0]
    print(f"The atomic number of the element is {atomic_number}.")

def draw_hangman(attempts, excited=False):
    stages = [
        # final state: excited hangman
        r"""
           |
           |       \ 0 /
           |        \|/
           |         |
           |         |
           |--------/-\----------
        """,
```

```
        # final state: head, torso, both arms, and both legs
        r"""
           --------
           |      |
           |      O
           |     \|/
           |      |
           |     / \
          ---
        """,
        # head, torso, both arms, and one leg
        r"""
           --------
           |      |
           |      O
           |     \|/
           |      |
           |     /
          ---
        """,
        # head, torso, and both arms
        r"""
           --------
           |      |
           |      O
           |     \|/
           |      |
           |
          ---
        """,
        # head, torso, and one arm
        r"""
           --------
           |      |
           |      O
           |     \|
           |      |
           |
          ---
        """,
        # head and torso
        r"""
           --------
           |      |
           |      O
           |      |
           |      |
           |
          ---
        """,
```

```
        # head
        r"""
           --------
           |      |
           |      O
           |
           |
           |
          ---
        """,
        # initial empty state
        r"""
           --------
           |      |
           |
           |
           |
           |
          ---
        """
    ]
    if excited:
        print(stages[0])
    else:
        print(stages[attempts])


def hangman(player):
    while True:
        print("Welcome to Hangman - Periodic Table Edition!")
        print("Try to guess the name of the chemical element.")
        print("You have 6 attempts to guess the word correctly.\n")

        df = load_periodic_table()
        difficulty = input("Select difficulty level (easy/medium/hard):
").lower()
        element_row = select_random_element(df, difficulty=difficulty)
        element_name = element_row['Name'].values[0]
        symbol = element_row['Symbol'].values[0]
        atomic_mass = element_row['AtomicMass'].values[0]
        nature = element_row['Nature'].values[0]
        category = element_row['AtomicNumber'].values[0]
        word = element_name.lower()

        displayed_word = display_word(word)

        attempts = 6
        guessed_letters = []
        start_time = time.time()
        total_time = 30  # Total time allowed for the game in seconds


        while attempts > 0 and '_ ' in displayed_word:
            elapsed_time = time.time() - start_time
            remaining_time = total_time - elapsed_time
```

```
            if remaining_time <= 0:
                print("\nTime's up!")
                break  # End the game if time is up

            print("\nWord to guess:", displayed_word)
            print("Attempts left:", attempts)
            print("Time left: {:.2f} seconds".format(remaining_time))
            draw_hangman(attempts)

            guess = input("Guess a letter: ").strip().lower()

            if guess == "hint":
                get_hint(element_row)
                continue

            if len(guess) != 1 or not guess.isalpha():
                print("Please enter a single letter.")
                continue
            elif guess in guessed_letters:
                print("You already guessed that letter.")
                continue

            guessed_letters.append(guess)

            if guess in word:
                displayed_word = update_displayed_word(word, guess,
displayed_word)
                print("Correct!")
                if '_ ' not in displayed_word:
                    print("Excited Hangman!")
                    draw_hangman(attempts, excited=True)
            else:
                print("Incorrect!")
                attempts -= 1

            elapsed_time = time.time() - start_time
            if elapsed_time > 120:
                print("\nTime's up!")
                break

        if '_ ' not in displayed_word:
            print("\nCongratulations! You guessed the word:", element_name)
            print("Symbol:", symbol)
            print("Atomic Number:", category)
            print("Atomic Mass:", atomic_mass)
            print("Nature:", nature)
        else:
            print("\nYou ran out of attempts or time. The word was:",
element_name)
            print("Symbol:", symbol)
            print("Atomic Number:", category)
            print("Atomic Mass:", atomic_mass)
            print("Nature:", nature)

        play_again = input("Do you want to play again? (yes/no):
").strip().lower()
```

```
        print()
        if play_again != 'yes':
            print("Thank you for playing! Keep studying, keep playing!")
            break



def multiplayer(players):
    print("\nLet's begin the multiplayer game!\n")
    for i, player in enumerate(players):
        print(f"It's {player}'s turn.")
        hangman(player)
        if i < len(players) - 1:
            print("\nNext player's turn.")

def main():
    while True:
        print("Welcome to Hangman - Periodic Table Edition!")
        print('**********************************************')
        print("Select an option:")
        print("1. Single Player")
        print("2. Multiplayer")
        choice = input("Enter your choice: ")

        if choice == '1':
            player = input("Enter your name: ").capitalize()  # Capitalize the
first letter of the name
            greet_player(player)
            hangman(player)
            break
        elif choice == '2':
            num_players = int(input("Enter the number of players: "))
            players = [input(f"Enter player {i+1}'s name: ").capitalize() for i
in range(num_players)]  # Capitalize the first letter of each player's name
            for player in players:
                greet_player(player)
            multiplayer(players)
            break
        else:
            print("Invalid choice. Please enter either 1 or 2.")
def greet_player(player):
    print(f"Hello, {player}! Let's play Hangman game.\n")

if __name__ == "__main__":
    main()
```

REFERENCES

1. https://en.wikipedia.org/wiki/Pandas_(software)
2. https://en.wikipedia.org/wiki/Comma-separated_values
3. https://en.wikipedia.org/wiki/Python_(programming_language)
4. IDLE - python coding platform
5. Github