# Python: Day 04

Excel Automation

## Previous Agenda

01

### Introduction

Objects and Classes

02

### Key Concepts

Concept Organization

03

### Maintainability

Development Guidelines

04

### Lab Session

Culminating Exercise

## Current Agenda

01

**OpenPyXL**

File-Based Handling

02

**Xlwings**

COM Introduction

03

**Pandas**

Dataframe Handling

04

**Win32Com**

Window-Based

05

**Streamlit**

Web App

06

**Lab Session**

Project Building

01

## OpenPyXL

Lightweight library for reading xlsx and xlsm files

# Excel Basics

Common Read-Write Operations for Excel Files

## Creating a Workbook

In OpenPyXL, an entire Excel file is represented using the **Workbook** class. All of the data processes (loading, saving, editing), sheet handling, and cell management is done here.

```
from openpyxl import Workbook

workbook = Workbook()



workbook.save("sample.xlsx")
```

## Default Worksheet

Accessing a worksheet is done using indexing. By default, a new workbook has a starting sheet with the title "Sheet"

```
from openpyxl import Workbook

workbook = Workbook()
sheet = workbook["Sheet"]


workbook.save("sample.xlsx")
```

## Creating a Worksheet

A `Workbook` object can use the `create_sheet(str)` method to create a new sheet. It gets added at the end by default. If you want to set the index, use `create_sheet(str, int)`.

```python
from openpyxl import Workbook

workbook = Workbook()
sheet = workbook["Sheet"]
workbook.create_sheet("Additional")

workbook.save("sample.xlsx")
```

## Editing a Cell

Accessing a worksheet is done using indexing. The key depends on the coordinate used in Excel workbooks

```python
from openpyxl import Workbook

workbook = Workbook()
sheet = workbook["Sheet"]
workbook.create_sheet("Additional")
sheet["A1"] = "Hello"
workbook.save("sample.xlsx")
```

## Loading a Worksheet

You can also load existing Excel files using the `load_workbook` helper function.

```python
from openpyxl import load_workbook

workbook = load_workbook("sample.xlsx")
```

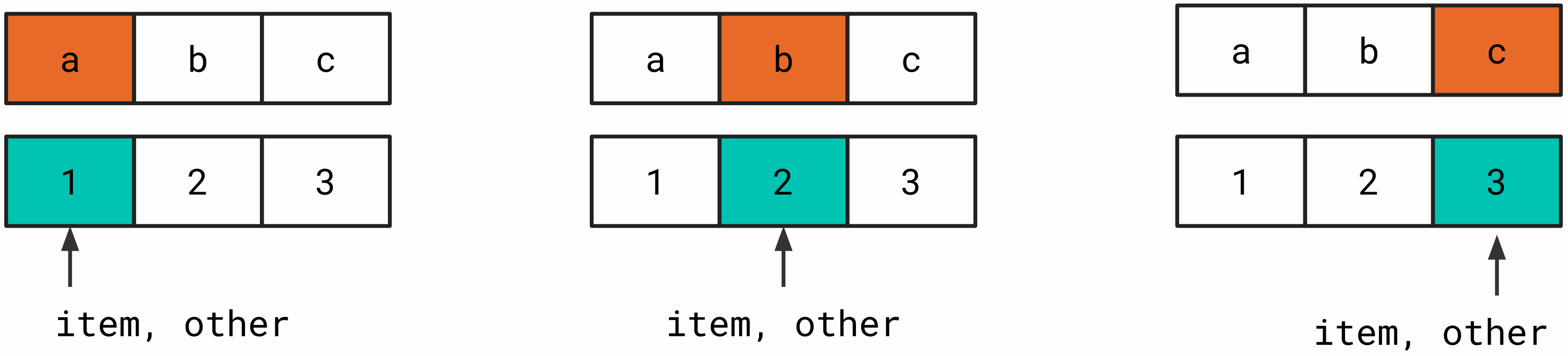
## Recap: Multi-Loop

Recall the mechanics of zip, enumerate, and tuple

## Multiple Looping

You can access two items at once from two different sequences using the zip function

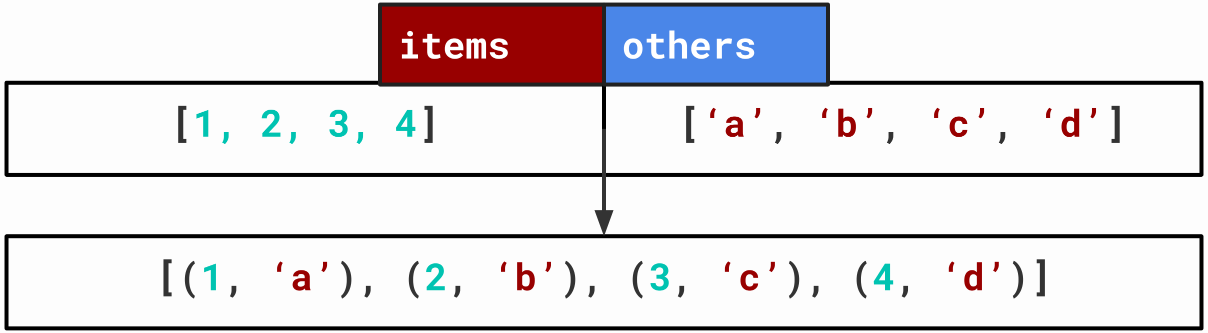
```
items = ('a', 'b', 'c')
others = (1, 2, 3)
for item, other in zip(items, others):
    print(item, other)
```



## Zip Function Contents

The **`zip`** function creates a list of tuples from all of its parameters

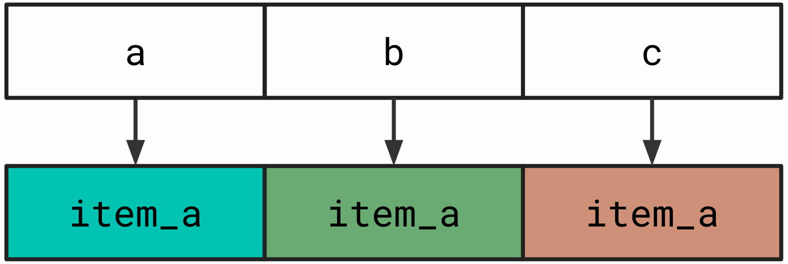
```
items = ('a', 'b', 'c')
others = (1, 2, 3)
zipped = zip(items, others)
print(list(zipped))
```



## Tuple Unpacking

Because tuples have a fixed size, Python added an unpacking feature for convenience

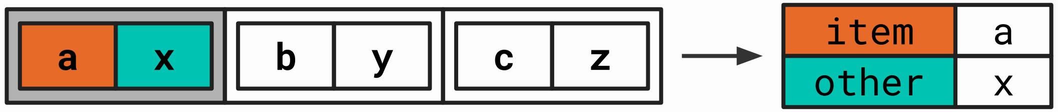
```python
items = ('a', 'b', 'c')
item_a, item_b, item_c = items
```



## Unpacking in Loops

You can access two items at once from two different sequences using the zip function

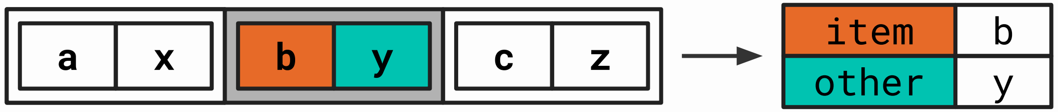
```
items = ('a', 'b', 'c')
others = ('x', 'y', 'z')
for item, other in zip(items, others):
    print(item, other)
```



## Unpacking in Loops

You can access two items at once from two different sequences using the zip function

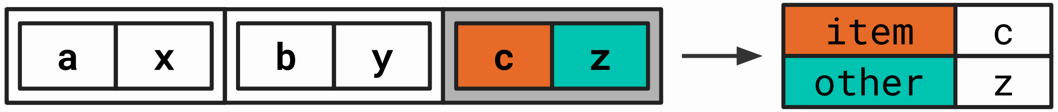
```python
items = ('a', 'b', 'c')
others = ('x', 'y', 'z')
for item, other in zip(items, others):
     print(item, other)
```



## Unpacking in Loops

You can access two items at once from two different sequences using the zip function

```python
items = ('a', 'b', 'c')
others = ('x', 'y', 'z')
for item, other in zip(items, others):
    print(item, other)
```



## Enumerate Looping

You can loop through a sequence of items and get their position using the enumerate function.

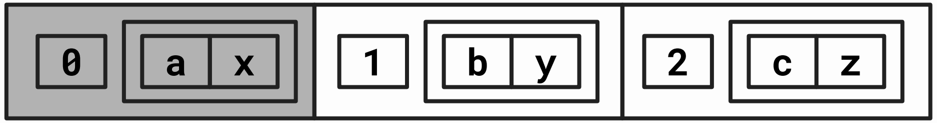
```
items = ('a', 'b', 'c')
for index, item in enumerate(items):
    print(index, item)
```

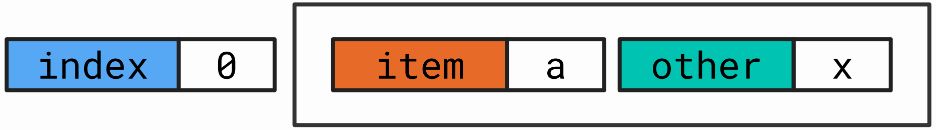
```
0 a
1 b
2 c
```

## Nested Unpacking

For inner tuples inside another tuple, denote using parentheses

```
items = ('a', 'b', 'c')
others = ('x', 'y', 'z')
for index, (items, other) in enumerate(zip(items, others)):
    print(item, other)
```

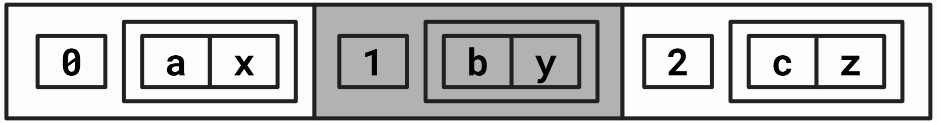


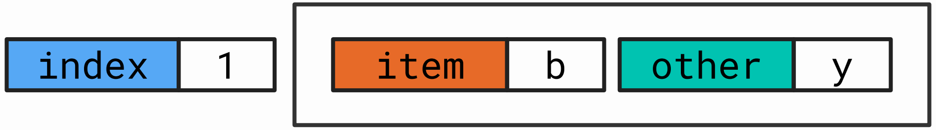


# Nested Unpacking

For inner tuples inside another tuple, denote using parentheses

```
items = ('a', 'b', 'c')
others = ('x', 'y', 'z')
for index, (items, other) in enumerate(zip(items, others)):
    print(item, other)
```

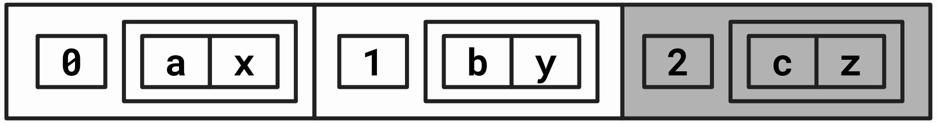


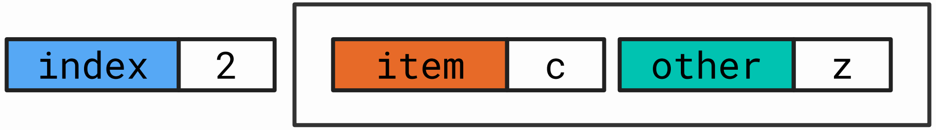


## Nested Unpacking

For inner tuples inside another tuple, denote using parentheses

```
items = (‘a’, ‘b’, ‘c’)
others = (‘x’, ‘y’, ‘z’)
for index, (items, other) in enumerate(zip(items, others)):
    print(item, other)
```





## Pair Unpacking

For inner tuples inside another tuple, denote using parentheses

```
dict1 = {'a': 1, 'b': 2}
dict2 = {'a': 10, 'b': 20}

for (k1, v1), (k2, v2) in zip(dict1.items(), dict2.items()):
    print(k1, v1, k2, v2)
```

## Cell Management

Example operations and methods for cell read and writes

## Read-Write Cells

Cells inside worksheets can either be accessed using indexing or the **Cell** interface.

```python
from openpyxl import load_workbook

workbook = load_workbook("sample.xlsx")
sheet = workbook["Additional"]

sheet["A1"] = "Tickets"
print(sheet["A1"].value)

cell = sheet.cell(row=1, column=2)
cell.value = 100
print(cell.value)

workbook.save("sample.xlsx")
```

## Multiple Cell Write

There is no dedicated method for writing in multiple cells at once. Instead, the expected approach is to use a standard loop

```python
from openpyxl import load_workbook

workbook = load_workbook("sample.xlsx")
sheet = workbook["Additional"]

tickets = {"HR": 30, "Legal": 23, "Sales": 34, "Admin": 13}

for i, (group, count) in enumerate(tickets.items(), start=3):
    sheet.cell(row=i, column=1).value = group
    sheet.cell(row=i, column=2).value = count

workbook.save("sample.xlsx")
```

## Multiple Cell Write (Ranges)

Worksheets support Excel-based formulas for getting items. This allows cell-based coding.

```python
from openpyxl import load_workbook

workbook = load_workbook("sample.xlsx")
sheet = workbook["Additional"]

tickets = {"HR": 30, "Legal": 23, "Sales": 34, "Admin": 13}

ticket_and_cells = zip(tickets.items(), sheet["A3:B6"])

for (group, count), (group_cell, count_cell) in ticket_and_cells:
    group_cell.value = group
    count_cell.value = count

workbook.save("sample.xlsx")
```

## Multiple Cell Append

While OpenPyXL doesn’t support writing on ranges directly, it allows appends.

```python
from openpyxl import load_workbook
workbook = load_workbook("sample.xlsx")
sheet = workbook["Additional"]

new_data = ["Tech", 300]
sheet.append(new_data)

workbook.save("sample.xlsx")
```

## Multiple Cell Read

Each **Worksheet** object has an `iter_rows` method to loop or iterate through all of the cells. Each row is a tuple of **Cell** objects.

```python
from openpyxl import load_workbook

workbook = load_workbook("sample.xlsx")
sheet = workbook["Additional"]

for row in sheet.iter_rows():
    print(row)
```

## Multiple Cell Read (Unpacked)

If there are only a few number of columns, you can directly assign the values to variables similar to how **enumerate** and **zip** operates.

```python
from openpyxl import load_workbook

workbook = load_workbook("sample.xlsx")
sheet = workbook["Additional"]

for header, item in sheet.iter_rows():
    print(header.value, item.value)
```

## Multiple Cell Read (Bounded)

The **`iter_rows`** method can change where it starts and ends using the min_row, and max_col optional parameters. The default is the first row and the last row with a value.

```
from openpyxl import load_workbook

workbook = load_workbook("sample.xlsx")
sheet = workbook["Additional"]

for header, item in sheet.iter_rows(min_row=3, max_row=6):
    print(header.value, item.value)
```

```
tip: you can use sheet.max_row and max.column
```

## Quick Exercise: Product Orders

Create a new sheet called **`Order`** in **`samples.xlsx`** and generate the following data

| **Category** | **Brand** | **Unit** |
|---|---|---|
| ***Laptop*** | HP | 1 |
| ***Laptop*** | HP | 2 |
| ***Laptop*** | Acer | 3 |
| ***Laptop*** | Acer | 4 |
| ***Monitor*** | HP | 1 |
| ***Monitor*** | HP | 2 |
| ***Monitor*** | Acer | 3 |
| ***Monitor*** | Acer | 4 |

## Cell+

Adding styling and rules for the cell layouts

## Cell Font

`Cell` objects have the `font` property that can be changed to add font-specific styling

```python
from openpyxl import load_workbook
from openpyxl.styles import Font

workbook = load_workbook("sample.xlsx")
sheet = workbook["Additional"]

sheet["A1"].font = Font(name="Arial", size=20)
workbook.save("sample.xlsx")
```

# Cell Font (Options)

`Cell` objects have the `font` property that can be changed to add styling

| Property | Description |
|---|---|
| name | `‘Calibri’, ‘Arial’, ‘Times New Roman’, etc. (system-based)` |
| size | `float/int` |
| bold | `bool` |
| italic | `bool` |
| underline | `‘single’, ‘double’, ‘singleAccounting’, ‘doubleAccounting’, None/False` |
| strike | `bool` |
| color | **`Hex Codes:`** `‘FF0000’ (Red), ‘00FF00’ (Green), ‘000000’ (Black), etc.` |

## Cell Pattern Fill

`Cell` objects have the `fill` property that can be changed to add background styling

```python
from openpyxl import load_workbook
from openpyxl.styles import PatternFill

workbook = load_workbook("sample.xlsx")
sheet = workbook["Additional"]

for (cell,) in sheet["A3:A7"]:
    cell.fill = PatternFill(fill_type='solid', fgColor='4F81BD')

workbook.save("sample.xlsx")
```

## Cell Pattern Border and Side

`Cell` objects have the `border` property that can be changed to add border styling

```python
from openpyxl import load_workbook
from openpyxl.styles import Side, Border

workbook = load_workbook("sample.xlsx")
sheet = workbook["Additional"]

ss = Side(style="thin", color='000000')

for (cell,) in sheet["A3:A7"]:
    cell.border = Border(left=ss, right=ss, top=ss, bottom=ss)

workbook.save("sample.xlsx")
```

## Cell Side (Options)

`Side` objects have the following styles to choose from

| Property | Description |
|---|---|
| style | `'thin', 'medium', 'thick', 'dashed', 'dotted', 'double', 'hair', 'mediumDashed', 'slantDashDot'` |
| color | **`Hex Codes:`** `‘FF0000’ (Red), ‘00FF00’ (Green), ‘000000’ (Black), etc.` |

## Cell Alignment

`Cell` objects have the `alignment` property that can be changed for text formatting

```python
from openpyxl import load_workbook
from openpyxl.styles import Alignment

workbook = load_workbook("sample.xlsx")
sheet = workbook["Additional"]

for (cell,) in sheet["A3:A7"]:
    cell.alignment = Alignment(
        horizontal='center', vertical='center',
        wrap_text=True, shrink_to_fit=True,
        indent=1
    )

workbook.save("sample.xlsx")
```

## Cell Alignment (Options)

The properties in the `Alignment` class have the following options

| Property | Description |
|---|---|
| horizontal | `'left', 'right', 'center', 'justify'` |
| vertical | `'top', 'center', 'bottom'` |

## Cell Number Format

`Cell` objects have the `alignment` property that can be changed for text formatting

```python
from openpyxl import load_workbook

workbook = load_workbook("sample.xlsx")
sheet = workbook["Additional"]

sheet["B1"].number_format = '#,##0'
workbook.save("sample.xlsx")
```

| | |
|---|---|
| Date Format | `'mm/dd/yyyy'` |
| Time | `'hh:mm:ss'` |
| Percentage | `'0%'` |
| Decimal | `'0.00'` |

## Quick Exercise: Product Orders (Styled)

Follow the styling below for the `Order` sheet in `samples.xlsx`

| **Category** | **Brand** | **Unit** |
|---:|---:|---:|
| ***Laptop*** | HP | 1 |
| ***Laptop*** | HP | 2 |
| ***Laptop*** | Acer | 3 |
| ***Laptop*** | Acer | 4 |
| ***Monitor*** | HP | 1 |
| ***Monitor*** | HP | 2 |
| ***Monitor*** | Acer | 3 |
| ***Monitor*** | Acer | 4 |

## Protection

Adding write safety to the worksheet

## Sheet Protection (Specific)

```python
from openpyxl import load_workbook


workbook = load_workbook("sample.xlsx")
sheet = workbook["Additional"]
sheet.protection.sheet = True




workbook.save("secured.xlsx")
```

## Sheet Protection (Specific)

```python
from openpyxl import load_workbook
from openpyxl.styles import Protection

workbook = load_workbook("sample.xlsx")
sheet = workbook["Additional"]
sheet.protection.sheet = True

for (cell,) in sheet["B2:B7"]:
    cell.protection = Protection(locked=False)

workbook.save("secured.xlsx")
```

## Data Validation (Contains)

Category-based (finite type of strings) can be limited using the `DataValidation` class

```python
from openpyxl import load_workbook
from openpyxl.worksheet.datavalidation import DataValidation

workbook = load_workbook("sample.xlsx")
sheet = workbook["Order"]

options_str = '"Laptop,Monitor,Peripheral"'
dv = DataValidation(type="list", formula1=options_str)

sheet.add_data_validation(dv)
dv.add("A2:A100")
workbook.save("sample.xlsx")
```

## Deletion

How to remove or clear out values

## Sheet Deletion

Remove a sheet can be done directly using the **`del`** operator

```python
from openpyxl import load_workbook

workbook = load_workbook("sample.xlsx")
del workbook["Sheet"]

workbook.save("sample.xlsx")
```

## Cell Deletion

There is no direct way to delete cells since it works on a reference basis but you can clear it

```
from openpyxl import load_workbook

workbook = load_workbook("sample.xlsx")
sheet = workbook["Additional"]
sheet["A1"] = None
sheet["B1"] = None

workbook.save("sample.xlsx")
```

## Row Deletion

There is no direct way to delete cells since it works on a reference basis but you can clear it

```
from openpyxl import load_workbook

workbook = load_workbook("sample.xlsx")
sheet = workbook["Additional"]
sheet.delete_rows(1)
sheet.delete_rows(1)

workbook.save("sample.xlsx")
```

H1

## Random Tickets

Generating random data for sanity checks

## Dummy Logs

Create a new workbook **`tickets.xlsx`**. In sheet **`Tickets`**, create `10_000` random entries

```
from random import randint, choice, seed
from datetime import datetime, timedelta

seed(123)

# Example of how to generate random values for a row
status = choice(["New", "Ongoing", "Done", "Close", None])
priority = choice(["Low", "Medium", "High", None])
department = choice(["HR", "Legal", "sales ", "Adm", "Tech"])
points = randint(1, 100)
votes = randint(1, 10)
start = datetime(2023, 5, 1) + timedelta(hours=randint(0, 2000))
end = start + timedelta(hours=randint(0, 2000))
```

02

## XLWings

Python package designed to call Excel operations and methods

## Creating a Workbook

In XLWings, an entire Excel file is represented using the **Book** class. All of the data processes are done here. Note that every **Book** stays open after running the code so make sure to close.

```python
import xlwings as xw

workbook = xw.Book()





workbook.save("sample_xl.xlsx")
workbook.close()
```

## Default Worksheet

Accessing a worksheet is done using indexing. Note that it needs to come from the `sheets` property. By default, a new workbook has a starting sheet with the title **"Sheet1"**

```python
import xlwings as xw

workbook = xw.Book()
sheet = workbook.sheets["Sheet1"]



workbook.save("sample_xl.xlsx")
workbook.close()
```

## Creating a Worksheet

A Book object has a sheets property which has the method add(str) to create a new sheet. It gets added at the end by default.

```
import xlwings as xw

workbook = xw.Book()
sheet = workbook.sheets["Sheet1"]
workbook.sheets.add("Additional")


workbook.save("sample_xl.xlsx")
workbook.close()
```

## Editing a Cell

Accessing a worksheet is done using indexing. Note that the keys are based on the coordinate system used by Excel workbooks.

```python
import xlwings as xw

workbook = xw.Book()
sheet = workbook.sheets["Sheet1"]
workbook.sheets.add("Additional")
sheet["A1"].value = "Hello"

workbook.save("sample_xl.xlsx")
workbook.close()
```

## Loading a Worksheet

You can load existing Excel files by using the filename when creating a **Book** object

```python
import xlwings as xw

workbook = xw.Book("sample_xl.xlsx")
workbook.close()
```

## Cell Management

Example operations and methods for cell read and writes

## Read-Write Cells

Cells inside worksheets can either be accessed using direct indexing or the range interface

```python
import xlwings as xw

workbook = xw.Book("sample_xl.xlsx")
sheet = workbook.sheets["Additional"]

sheet["A1"].value = "Tickets"
print(sheet["A1"].value)

cell = sheet.range("B1")
cell.value = 100
print(cell.value)

workbook.save("sample_xl.xlsx")
workbook.close()
```

## Multiple Cell Write

One of the key features of xlwings is that it supports assignment for range objects to enable bulk writing

```
import xlwings as xw

workbook = xw.Book("sample_xl.xlsx")
sheet = workbook["Additional"]

tickets = {"HR": 30, "Legal": 23, "Sales": 34, "Admin": 13}
sheet.range("A3:B6").value = list(tickets.items())

workbook.save("sample_xl.xlsx")
workbook.close()
```

## Multiple Cell Read

The range interface of the Book sheets interface can be directly iterated and unpacked

```python
import xlwings as xw

workbook = xw.Book("sample_xl.xlsx")
sheet = workbook.sheets["Additional"]

for header, item in sheet.range("A1:B7").value:
    print(header, item)
```

## Quick Exercise: Product Orders

Create a new sheet called **`Order`** in **`samples.xlsx`** and generate the following data

| **Category** | **Brand** | **Unit** |
|---|---|---|
| ***Laptop*** | HP | 1 |
| ***Laptop*** | HP | 2 |
| ***Laptop*** | Acer | 3 |
| ***Laptop*** | Acer | 4 |
| ***Monitor*** | HP | 1 |
| ***Monitor*** | HP | 2 |
| ***Monitor*** | Acer | 3 |
| ***Monitor*** | Acer | 4 |

## Cell+

Adding styling and rules for the cell layouts

## Cell Font

**Range** interfaces have the **Font** property that have further attributes to change for styling

```python
import xlwings as xw

workbook = xw.Book("sample_xl.xlsx")
sheet = workbook.sheets["Additional"]

cell = sheet.range("A1")
cell.api.Font.Name = "Calibri"
cell.api.Font.Size = 14
cell.api.Font.Bold = True
cell.api.Font.Italic = True
cell.api.Font.Underline = True # Examples: True, False, 2
cell.api.Font.Color = 0x0000FF

workbook.save("sample_xl.xlsx")
workbook.close()
```

## Cell Background Color

**Range** interfaces also the `Interior` property, often used to change the cell color

```python
import xlwings as xw

workbook = xw.Book("sample_xl.xlsx")
sheet = workbook.sheets["Additional"]

cell = sheet.range("A1")
cell.api.Interior.Color = 0xFFFF00

workbook.save("sample_xl.xlsx")
workbook.close()
```

## Cell Pattern Border and Side

**Range** interfaces have the `Borders` property to add border styling

```python
import xlwings as xw

workbook = xw.Book("sample_xl.xlsx")
sheet = workbook.sheets["Additional"]

cell = sheet.range("A1")
cell.api.Borders(9).LineStyle = 1
cell.api.Borders(9).Weight = 3
cell.api.Borders(9).Color = 0x000000

workbook.save("sample_xl.xlsx")
workbook.close()
```

## Border Types

`Border` objects follow from the constants used in the COM interface by Windows

| VBA Constant | Value | Border Type |
|---|---|---|
| xlEdgeLeft | 7 | `Left Border` |
| xlEdgeTop | 8 | `Top Border` |
| xlEdgeBottom | 9 | `Bottom Border` |
| xlEdgeRight | 10 | `Right Border` |
| xlInsideVertical | 11 | `Inner Vertical` |
| xlInsideHorizontal | 12 | `Inner Horizontal` |

## Line Styles

`LineStyle` objects follow from the constants used in the COM interface by Windows

| VBA Constant | Value | Border Type |
|---|---|---|
| xlContinuous | 1 | Solid line (most common) |
| xlDash | -4115 | Dashed line |
| xlDashDot | 4 | Dash-dot line |
| xlDashDotDot | 5 | Dash-dot-dot line |
| xlDot | -4118 | Dotted line |
| xlLineStyleNone | -4142 | No border (removes it) |

## Cell Alignment

**Range** interfaces have alignment properties to change text layouts

```python
import xlwings as xw

workbook = xw.Book("sample_xl.xlsx")
sheet = workbook.sheets["Additional"]

cell = sheet.range("A1")
cell.api.HorizontalAlignment = -4108
cell.api.VerticalAlignment = -4108
cell.api.WrapText = True

workbook.save("sample_xl.xlsx")
workbook.close()
```

## Horizontal Alignment

`HorizontalAlignment` follow from the constants used in the COM interface by Windows

| VBA Constant | Value | Border Type |
|---|---|---|
| xlGeneral | 1 | Default (depends on value type) |
| xlLeft | -4131 | Align text to the left |
| xlCenter | -4108 | Center text horizontally |
| xlRight | -4152 | Align text to the right |
| xlJustify | -4130 | Justify text across the cell |
| xlDistributed | -4117 | Even spacing (requires wrap) |

## Vertical Alignment

`VerticalAlignment` follow from the constants used in the COM interface by Windows

| VBA Constant | Value | Border Type |
|---|---|---|
| xlTop | -4160 | `Align text to the top` |
| xlCenter | -4108 | `Center text vertically` |
| xlBottom | -4107 | `Align text to the bottom` |
| xlJustify | -4130 | `Justify vertically` |
| xlDistributed | -4117 | `Justify text across the cell` |

## Cell Number Format

`Cell` objects have the `alignment` property that can be changed for text formatting

```python
import xlwings as xw

workbook = xw.Book("sample_xl.xlsx")
sheet = workbook.sheets["Additional"]

sheet["B1"].number_format = '#,##0'
```

| | |
|---|---|
| Date Format | `'mm/dd/yyyy'` |
| Time | `'hh:mm:ss'` |
| Percentage | `'0%'` |
| Decimal | `'0.00'` |

## Quick Exercise: Product Orders (Styled)

Follow the styling below for the **`Order`** sheet in **`samples.xlsx`**

| Category | Brand | Unit |
|---|---|---|
| ***Laptop*** | HP | 1 |
| ***Laptop*** | HP | 2 |
| ***Laptop*** | Acer | 3 |
| ***Laptop*** | Acer | 4 |
| ***Monitor*** | HP | 1 |
| ***Monitor*** | HP | 2 |
| ***Monitor*** | Acer | 3 |
| ***Monitor*** | Acer | 4 |

## Deletion

How to remove or clear out values

## Sheet Deletion

Remove a sheet can be done directly using the `delete` method

```python
import xlwings as xw

workbook = xw.Book("sample_xl.xlsx")
workbook.sheets["Sheet1"].delete()

workbook.save("sample_xl.xlsx")
workbook.close()
```

## Range Deletes

Full ranges can be deleted with the COM interface

```python
import xlwings as xw

workbook = xw.Book("sample_xl.xlsx")
sheet = workbook.sheets["Additional"]

sheet.range("1:1").api.Delete()

workbook.save("sample_xl.xlsx")
workbook.close()
```

## Dynamic Actions

Handling actual Excel instances in code

## Calculate Refresh

For very large systems or when Excel does not have auto-calculate enabled, you can force it

```python
import xlwings as xw

workbook = xw.Book()
sheet = workbook.sheets[0]

sheet["A1"].formula = "=B1 + 10"
sheet["B1"].value = 5

workbook.app.calculate()

print(sheet["A1"].value)
workbook.close()
```

## Calling Macros

Macros can be accessed for Workbooks using the `macro` property and its name

```
Sub GreetUser()
    MsgBox "Hello from Excel VBA!"
End Sub
```

```python
import xlwings as xw

workbook = xw.Book("example.xlsm")
greet_macro = workbook.macro("GreetUser")
greet_macro()
```

H2

## Dummy Accounts

Generating random data for sanity checks later on

## Dummy Accounts

Create a new workbook **`accounts.xlsx`**. In sheet **`Logs`** create `10_000` random entries

```python
from random import randint, choice, seed
from datetime import datetime, timedelta

seed(123)

# Example of how to generate random values for a row
accounts = choice([...])
sector = choice([...])
year_established = randint(1900, 2025)
revenue = randint(10_000, 100_000_000_000)
employees = randint(1, 1_000_000)
office_location = choice([...])
subsidiary_of = choice([...])
```

## Dummy Accounts - Statistics

Additionally, make a new sheet **`Summary`** that has the follow key values

- Total Revenue
- Average Revenue
- Minimum Revenue
- Earliest Year Established
- Most Recent Year Established
- Most Common Sector

03

# Pandas

The most common technique for tabular data manipulation

## Reading Data

Pandas converts tabular data to data frames that are convenient to read and access

```python
import pandas as pd

df = pd.read_csv("tickets.csv")
print(df)
print(df.info())
print(df.describe())
```

```python
import pandas as pd

df = pd.read_excel("tickets.xlsx", sheet_name="Tickets")
print(df)
print(df.info())
print(df.describe())
```

## Dataframe Columns

Pandas makes column access very convenient using the indexing operation

```python
import pandas as pd

df = pd.read_excel("tickets.xlsx", sheet_name="Tickets")
print(df.columns)
print(df["Priority"])
print(df["Priority"].unique())
print(df["Priority"].value_counts())
```

## Dataframe New Columns

Pandas specializes in creating new columns using data from other columns

```python
import pandas as pd

df = pd.read_excel("tickets.xlsx", sheet_name="Tickets")

df["Duration"] = df["End"] - df["Start"]
df["Duration"] = df["Duration"].dt.total_seconds()
df["Duration"] = df["Duration"] / 3600

print(df)
```

## Data Processes

Common operations and methods for data preparation

## Common Data Cleaning Techniques

```python
import pandas as pd

df = pd.read_excel("tickets.xlsx", sheet_name="Tickets")
df.columns = df.columns.str.strip().str.title()

df["Department"] = df["Department"].str.strip().str.title()
df["Status"].fillna("Unknown", inplace=True)
df.dropna(subset=["Priority"], inplace=True)

print(df)
```

## Sorting by Column

```python
import pandas as pd

df = pd.read_excel("tickets.xlsx", sheet_name="Tickets")
df.columns = df.columns.str.strip().str.title()

df["Department"] = df["Department"].str.strip().str.title()
df["Status"].fillna("Unknown", inplace=True)
df.dropna(subset=["Priority"], inplace=True)

df.sort_values(
    by='year_established', ascending=False)

print(df)
```

## Saving in a New Excel File

```python
import pandas as pd

df = pd.read_excel("tickets.xlsx", sheet_name="Tickets")
df.columns = df.columns.str.strip().str.title()

df["Department"] = df["Department"].str.strip().str.title()
df["Status"].fillna("Unknown", inplace=True)
df.dropna(subset=["Priority"], inplace=True)

df.sort_values(
    by='year_established', ascending=False)

print(df)
df.to_excel("tick_new.xlsx", sheet_name="Tickets", index=False)
```

## Appending to an Existing Excel File

```python
import pandas as pd

df = pd.read_excel("tickets.xlsx", sheet_name="Tickets")
df.columns = df.columns.str.strip().str.title()

df["Department"] = df["Department"].str.strip().str.title()
df["Status"].fillna("Unknown", inplace=True)
df.dropna(subset=["Priority"], inplace=True)

df.sort_values(
    by='year_established', ascending=False)

print(df)
with pd.ExcelWriter('tickets.xlsx', mode='a') as writer:
    df.to_excel(writer, sheet_name="Clean Tickets", index=False)
```

## Pandas Filtering

```python
import pandas as pd

df = pd.read_excel("accounts.xlsx", sheet_name="Logs")

high_revenue = df[df['Revenue'] > 100_000_000]
tech_sector = df[df['Sector'] == "Technology"]

print(df)
with pd.ExcelWriter('accounts.xlsx', mode='a') as writer:
    tech_sector.to_excel(writer, sheet_name="Tech", index=False)
    high_revenue.to_excel(writer, sheet_name="Top", index=False)
```

## Grouping and Aggregation

```python
import pandas as pd

df = pd.read_excel("accounts.xlsx", sheet_name="Logs")

avg_revenue = df.groupby('Sector')['Revenue'].mean()
total_employees = df.groupby('Sector')['Employees'].sum()
sector_count = df['Sector'].value_counts()

print('Average Revenue', avg_revenue)
print('Total Employees', total_employees)
print('Sector Count', sector_count)
```

## Data Visualization

Examples of all visualizations

## Histogram (Number Distribution)

```python
import matplotlib.pyplot as plt
import pandas as pd

df = pd.read_excel("accounts.xlsx", sheet_name="Logs")
df["Revenue"].hist(bins=30, color="skyblue", edgecolor="black")
plt.title("Revenue Distribution")
plt.xlabel("Revenue")
plt.ylabel("Frequency")
plt.show()
```

## Bar Chart (Change Over Unit)

```python
import matplotlib.pyplot as plt
import pandas as pd

df = pd.read_excel("accounts.xlsx", sheet_name="Logs")
df["Sector"].value_counts().plot.bar(color="orange")
plt.title("Companies per Sector")
plt.xlabel("Sector")
plt.ylabel("Count")
plt.show()
```

## Scatter Plot Chart (Spatial Relationship)

```python
import matplotlib.pyplot as plt
import pandas as pd

df = pd.read_excel("accounts.xlsx", sheet_name="Logs")
df["Office Location"].value_counts().head(5).plot.pie()
plt.title("Top 5 Office Locations (Share)")
plt.xlabel("Sector")
plt.ylabel("")
plt.show()
```

## Pie Chart (Percent Composition)

```python
import matplotlib.pyplot as plt
import pandas as pd

df = pd.read_excel("accounts.xlsx", sheet_name="Logs")
df["Office Location"].value_counts().head(5).plot.pie()
plt.title("Top 5 Office Locations (Share)")
plt.xlabel("Sector")
plt.ylabel("")
plt.show()
```

## Box Plot (Statistics Summary)

```python
import matplotlib.pyplot as plt
import pandas as pd

df = pd.read_excel("accounts.xlsx", sheet_name="Logs")
df.boxplot(column="Revenue", by="Sector")
plt.title("Revenue Distribution by Sector")
plt.xlabel("Sector")
plt.ylabel("Revenue")
plt.tight_layout()
plt.show()
```

## Line Plot（Change Over Unit）

```python
import matplotlib.pyplot as plt
import pandas as pd

df = pd.read_excel("accounts.xlsx", sheet_name="Logs")
df.groupby("Year Established")["Revenue"].mean().plot.line()
plt.title("Average Revenue by Year Established")
plt.xlabel("Year")
plt.ylabel("Average Revenue")
plt.show()
```

## Stacked Bar Chart (Composition + Growth)

```python
import matplotlib.pyplot as plt
import pandas as pd

df = pd.read_excel("accounts.xlsx", sheet_name="Logs")
stack_data = df.groupby(["Year Established", "Sector"])
stack_data = stack_data.size().unstack().fillna(0)

stack_data.plot.bar(stacked=True)
plt.title("Companies per Year by Sector")
plt.xlabel("Year Established")
plt.ylabel("Company Count")
plt.tight_layout()
plt.show()
```

H3

# Data Analysis

An introduction to data science

## Data Analysis - Easy

Answer the following questions related to the data:

- How many accounts are there?
- How many columns are available?
- What is the average revenue?
- What is the average number of employees?
- How many companies are there per sector?
- What are the five most common office locations?

## Data Analysis - Intermediate

Answer the following questions related to the data:

- Which sector has the most accounts?
- Which sector has the highest total revenue?
- What is the average revenue per office location?
- What is the average number of employees?
- Which location has the highest employee count?
- What sectors have the oldest companies?

## Data Analysis - Advance

Answer the following questions related to the data:

- How many companies earn over 10 million?
  - What is their average employee count?
  - Is there a relationship between revenue and employee count?
- What are the top companies by revenue?
  - What sectors and locations do they belong to?
- What is the difference between the top sector and the worst sector?

04

## Win32 COM

Handling Windows Applications using Python

## Outlook Cell Edits

The COM interface is more exposed using the win32com Python Library

```python
import win32com.client as win32

excel = win32.Dispatch("Excel.Application")
workbook = excel.Workbooks.Add()
sheet = workbook.Sheets(1)

sheet.Cells(1, 1).Value = "Hello"
sheet.Cells(1, 1).Font.Bold = True
sheet.Range("A2:B2").Merge()
sheet.Range("A2").Value = "Merged Cell"
sheet.Range("A1:B2").Interior.Color = 0x00FF00
sheet.Range("A3").Value = "Width Test"
sheet.Columns("A:B").AutoFit()
sheet.Rows("1:3").RowHeight = 30
```

## Outlook Email Sending

The COM interface can also handle Outlook for sending emails

```python
import win32com.client as win32

outlook = win32.Dispatch("Outlook.Application")

mail = outlook.CreateItem(0)
mail.Subject = "Email Subject"
mail.Body = "Hello, this is a test email sent from Python!"
mail.To = "recipient@example.com"
mail.CC = "cc1@example.com; cc2@example.com"
mail.BCC = "bcc1@example.com; bcc2@example.com"

mail.Attachments.Add("accounts.xlsx")

mail.Display()
```

## Word Document Writing

```python
import os
import win32com.client as win32

def read_word_doc():
    file_path = os.path.join(os.getcwd(), "test.docx")

    word = win32.Dispatch("Word.Application")
    doc = word.Documents.Open(file_path)
    content = doc.Content.Text

    doc.Close()
    word.Quit()

    return content

print(read_word_doc())
```

## Outlook Email Sending (HTML)

```python
import win32com.client as win32

outlook = win32.Dispatch("Outlook.Application")
mail = outlook.CreateItem(0)

mail.To = "stephen.singer.098@gmail.com"
mail.Subject = "Email Subject"
mail.HTMLBody =  """
<html>
    <body>
        <h1>Hello, this is a test email sent from Python!</h1>
        <p>This is <strong>HTML</strong> content.</p>
        <p><a href="http://www.example.com">Click here</a> to visit.</p>
    </body>
</html>
"""
mail.Display()

```

## Outlook Email Reading

```python
import win32com.client as win32

outlook = win32.Dispatch("Outlook.Application")
namespace = outlook.GetNamespace("MAPI")
inbox = namespace.GetDefaultFolder(6)

messages = inbox.Items
messages.Sort("[ReceivedTime]", True)

for message in messages:
     if hasattr(message, 'Subject')
          if "Rep" in message.Subject:
               if hasattr(message, 'SenderEmailAddress'):
                    if message.SenderEmailAddress =="test@example.com":
                         print(message.Body)
                         break
```

H4

## Send Me an Email

Provide your short, genuine feedback with the bootcamp

## XLSXWriter

Easiest method for making charts

## Column Chart

```python
import xlsxwriter

wb = xlsxwriter.Workbook('column_chart.xlsx')
ws = wb.add_worksheet()

data = [10, 20, 30, 40]
ws.write_column('A1', data)

chart = wb.add_chart({'type': 'column'})
chart.add_series({'values': '=Sheet1!$A$1:$A$4'})
ws.insert_chart('C1', chart)

wb.close()
```

## Line Chart

```python
import xlsxwriter

wb = xlsxwriter.Workbook('line_chart.xlsx')
ws = wb.add_worksheet()

data = [5, 15, 10, 25]
ws.write_column('A1', data)

chart = wb.add_chart({'type': 'line'})
chart.add_series({'values': '=Sheet1!$A$1:$A$4'})
ws.insert_chart('C1', chart)

wb.close()
```

## Pie Chart

```python
import xlsxwriter

wb = xlsxwriter.Workbook('pie_chart.xlsx')
ws = wb.add_worksheet()

labels = ['Apples', 'Bananas', 'Cherries']
values = [30, 20, 50]
ws.write_column('A1', labels)
ws.write_column('B1', values)

chart = wb.add_chart({'type': 'pie'})
chart.add_series({
    'categories': '=Sheet1!$A$1:$A$3',
    'values': '=Sheet1!$B$1:$B$3'
})
ws.insert_chart('D1', chart)

wb.close()
```

## Scatter Chart

```python
import xlsxwriter

wb = xlsxwriter.Workbook('scatter_chart.xlsx')
ws = wb.add_worksheet()

x = [1, 2, 3, 4]
y = [10, 20, 30, 25]
ws.write_column('A1', x)
ws.write_column('B1', y)

chart = wb.add_chart({'type': 'scatter'})
chart.add_series({
    'categories': '=Sheet1!$A$1:$A$3',
    'values': '=Sheet1!$B$1:$B$3'
})
ws.insert_chart('D1', chart)

wb.close()
```

05

## Streamlit

Modern web app framework for simple, data-driven use cases

## Virtual Environments

Prerequisite for using Streamlit

## Virtual Environment

A virtual environment (venv) isolates packages for your project from the entire system. This prevents package conflicts, prevents clutter, and makes the project reproducible. The following code creates a folder .venv that will store isolated packages

**Windows**

```
$ python -m venv .venv
```

**Linux/MacOS**

```
$ python3 -m venv .venv
```

## Virtual Environment - Activation

To actually use the packages of a virtual environment, you need to **`activate`** it first.

**Windows (Command Prompt)**

```
$ .venv\Scripts\activate
```

**Windows (Powershell)**

```
$ .venv\Scripts\Activate.ps1
```

**Linux/MacOS**

```
$ source .venv/bin/activate
```

## Virtual Environment - Deactivation

To exit the virtual environment, simply enter **`deactivate`** on any console

```
$ deactivate
```

## GUI

Visually connecting and constraining the user

## Benefits of Graphical User Interfaces

### User Experience

Easier to understand and provides appeal and interactivity

### Separation

Clearly Separate Frontend (Design, UI/UX) and Backend (Logic)

### Limitations

Limit the possible edge cases and directly get needed data type

## Python GUI Libraries

### Tkinter

Standard GUI toolkit available in (almost) all Python distributions immediately. Easy to understand and great for building simple applications quickly.

### PyQt

Python bindings or implementations for the Qt application framework. It has a lot of flexible components and great for building complex applications.

### Kivy

Library built specifically for multi-touch platforms (mobile) but can be used in Desktops as well. Good for complex, cross-platform applications.

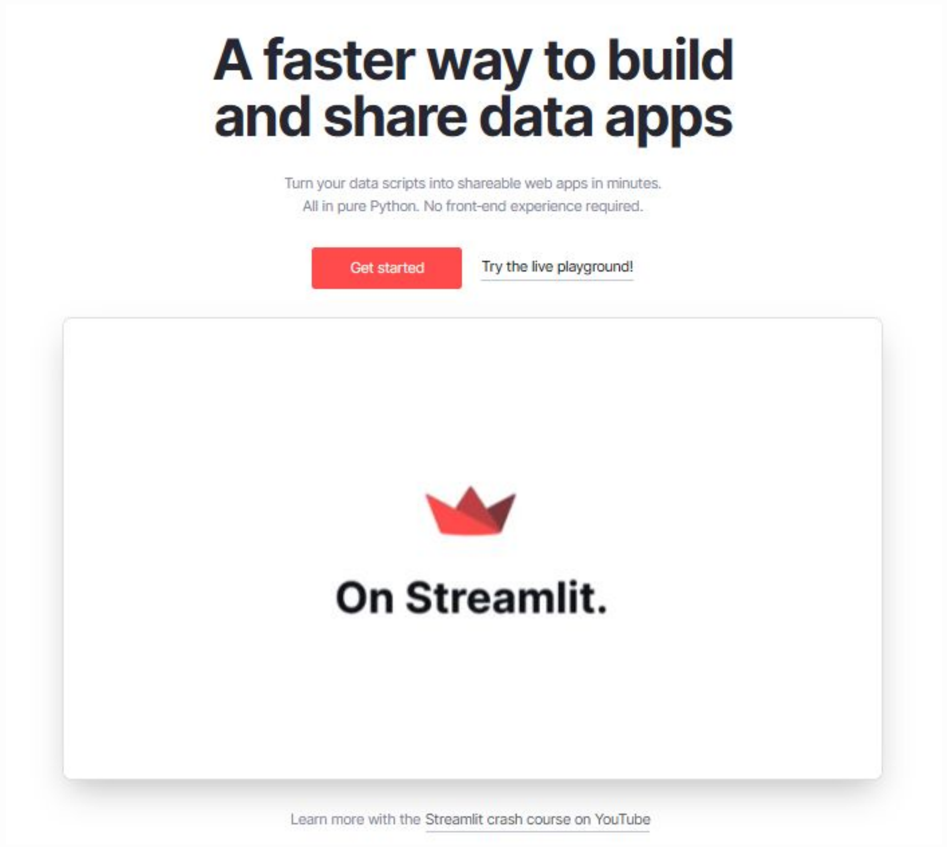
## Web Frameworks

### Flask

- Minimalist and lightweight
- Freedom to choose tools for each part
- **Small and Fast Web Applications**

### Django

- Multiple out-of-the-box features
  - Object Relational Mapping
  - Fully functional Admin Panel
  - Security Measures and Authentication
- **Medium to Large Web applications**



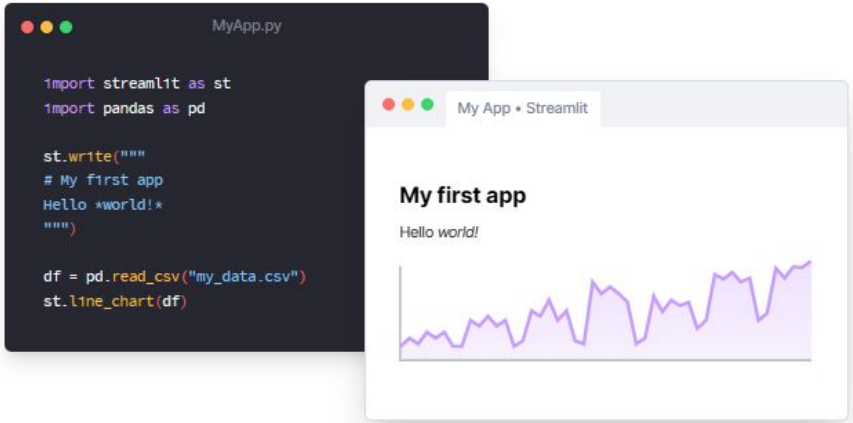
A faster way to build and share data apps

Turn your data scripts into shareable web apps in minutes.
All in pure Python. No front-end experience required.

Learn more with the Streamlit crash course on YouTube

## Embrace scripting

Build an app in a few lines of code with our magically simple API. Then see it automatically update as you iteratively save the source file.



MyApp.py

```
import streamlit as st
import pandas as pd

st.write("""
# My first app
Hello *world!*
""")

df = pd.read_csv("my_data.csv")
st.line_chart(df)
```

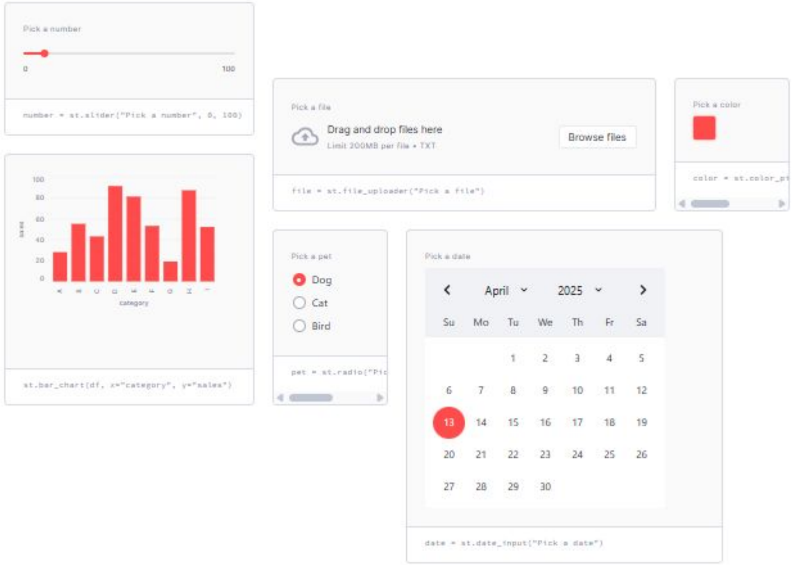
My App • Streamlit

My first app

*Hello world!*

## Weave in interaction

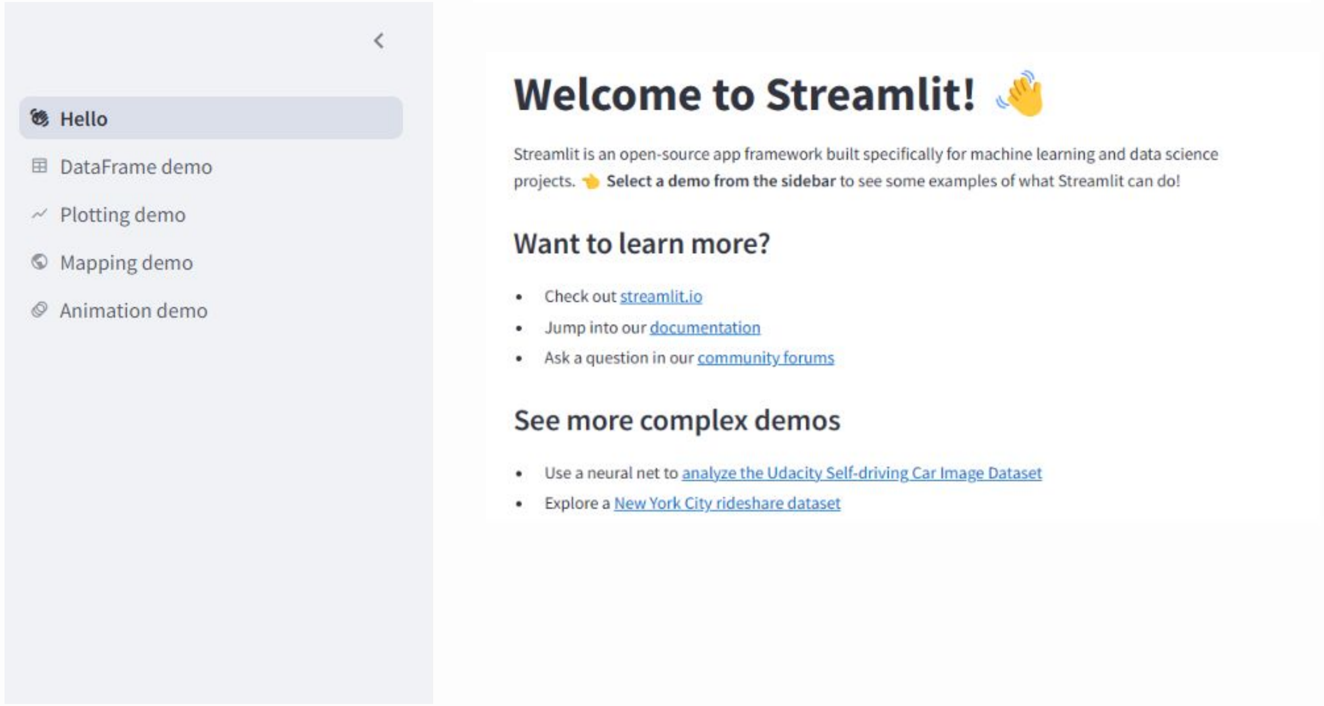
Adding a widget is the same as declaring a variable. No need to write a backend, define routes, handle HTTP requests, connect a frontend, write HTML, CSS, JavaScript, ...



```
number = st.slider("Pick a number", 0, 100)
file = st.file_uploader("Pick a file")
color = st.color_pi
st.bar_chart(df, x="category", y="sales")
pet = st.radio("Pic
date = st.date_input("Pick a date")
```

## Get started in under a minute

```
$ pip install streamlit
$ streamlit hello
```



Welcome to Streamlit! 👋

Streamlit is an open-source app framework built specifically for machine learning and data science projects. 👈 **Select a demo from the sidebar** to see some examples of what Streamlit can do!

Want to learn more?

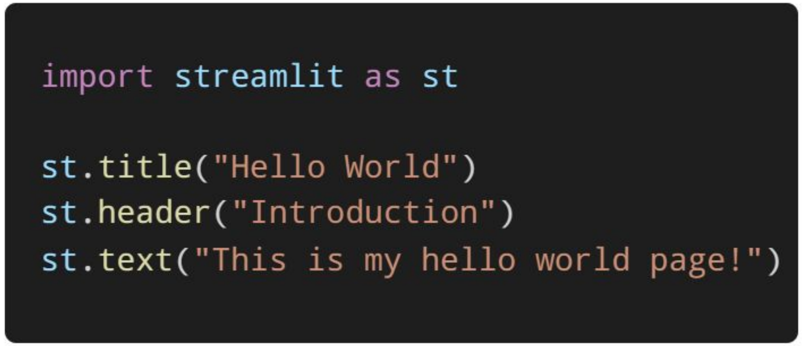
- Check out streamlit.io
- Jump into our documentation
- Ask a question in our community forums

See more complex demos

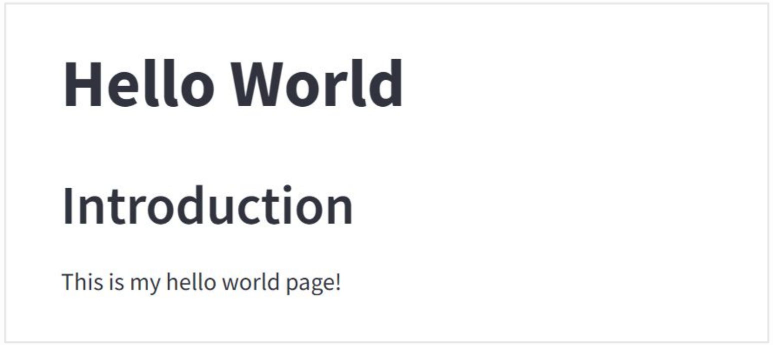
- Use a neural net to analyze the Udacity Self-driving Car Image Dataset
- Explore a New York City rideshare dataset

## Streamlit: Hello World

Make a new file with the following Python code.



```python
import streamlit as st

st.title("Hello World")
st.header("Introduction")
st.text("This is my hello world page!")
```



# Hello World

## Introduction

This is my hello world page!

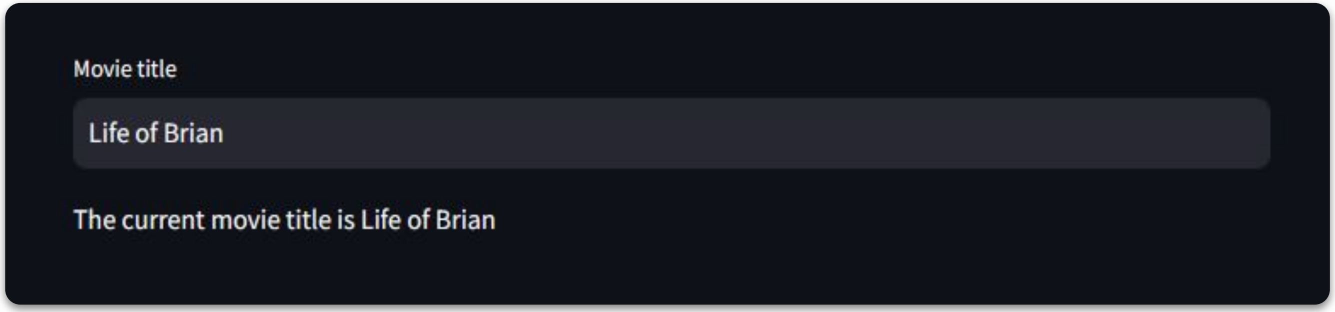
## Components

Learn some of the available interactive elements

## Text Input

The **`st.text_input`** displays a single-line text input widget.

```python
import streamlit as st

title = st.text_input("Movie title", "Life of Brian")
st.write("The current movie title is", title)
```



Movie title

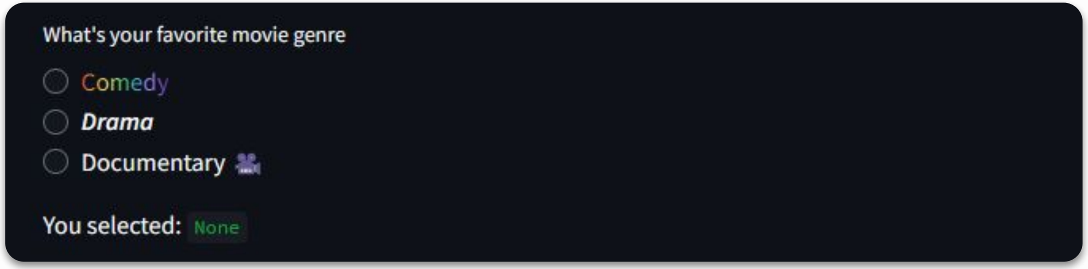
Life of Brian

The current movie title is Life of Brian

## Radio Buttons

The **`st.radio`** displays a radio button widget

```python
import streamlit as st

genre = st.radio(
    "What's your favorite movie genre",
    [":rainbow[Comedy]", "***Drama***", "Documentary :movie_camera:"],
    index=None,
)

st.write("You selected:", genre)
```



What's your favorite movie genre

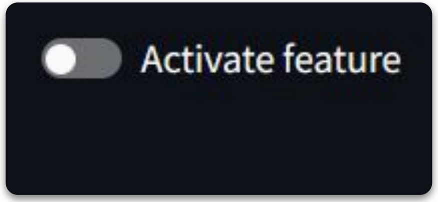
- Comedy
- ***Drama***
- Documentary 🎥

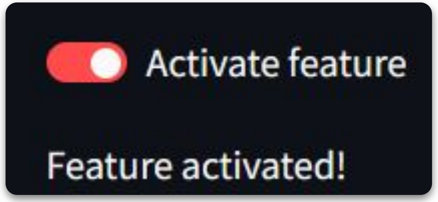
You selected: `None`

## Toggle

The **`st.toggle`** displays a slider widget for integers, time, and datetime values

```
import streamlit as st

on = st.toggle("Activate feature")

if on:
    st.write("Feature activated!")
```

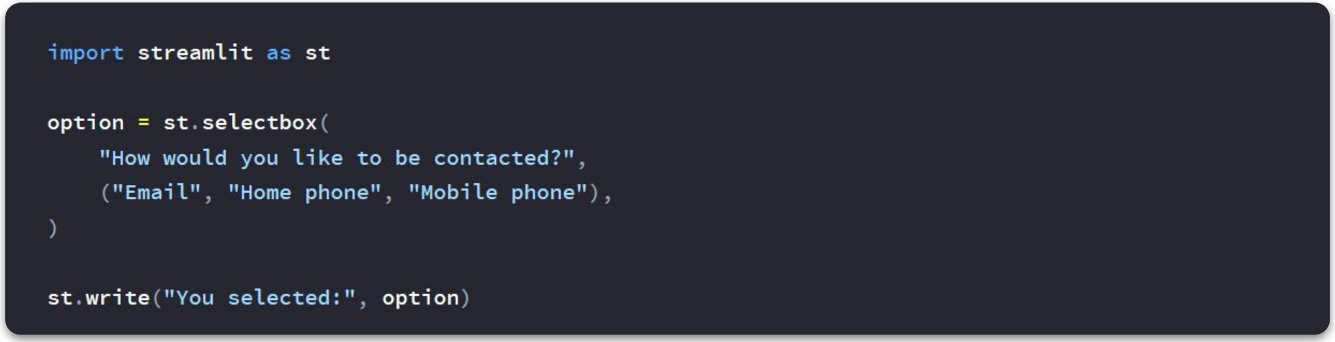

Activate feature


Activate feature

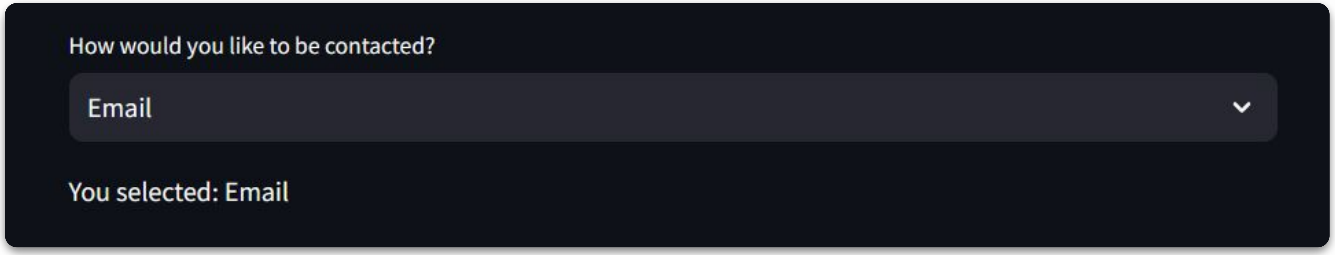
Feature activated!

## Select Box

The **`st.select_box`** displays a select widget for choosing a single value



```
import streamlit as st

option = st.selectbox(
    "How would you like to be contacted?",
    ("Email", "Home phone", "Mobile phone"),
)

st.write("You selected:", option)
```



How would you like to be contacted?

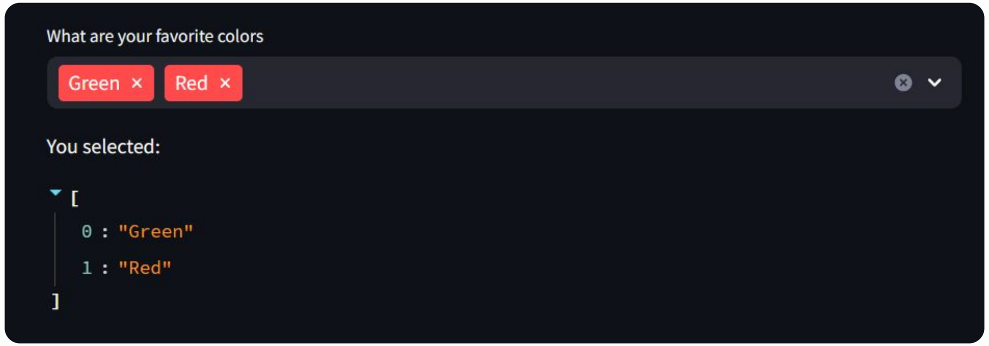
Email

You selected: Email

## Multiselect

The **`st.multiselect`** displays a multiselect widget

```python
import streamlit as st

options = st.multiselect(
    "What are your favorite colors",
    ["Green", "Yellow", "Red", "Blue"],
    ["Yellow", "Red"],
)

st.write("You selected:", options)
```



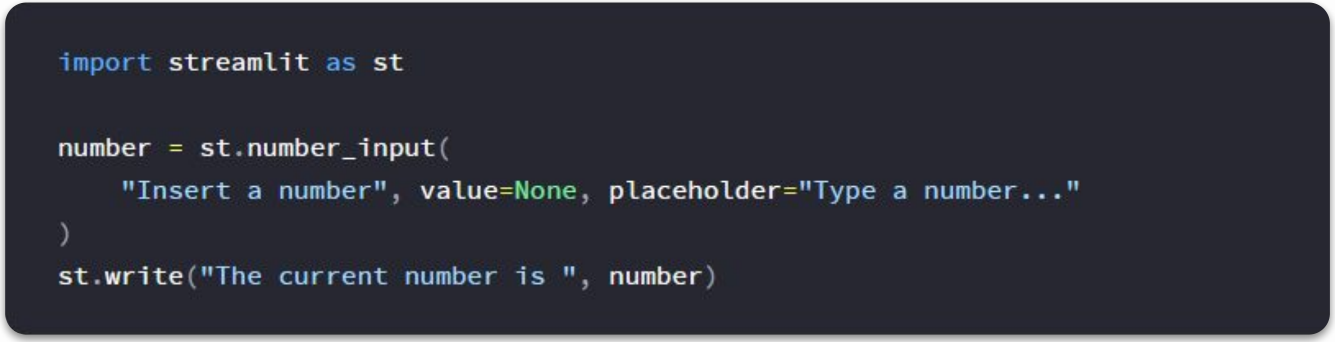
What are your favorite colors

You selected:

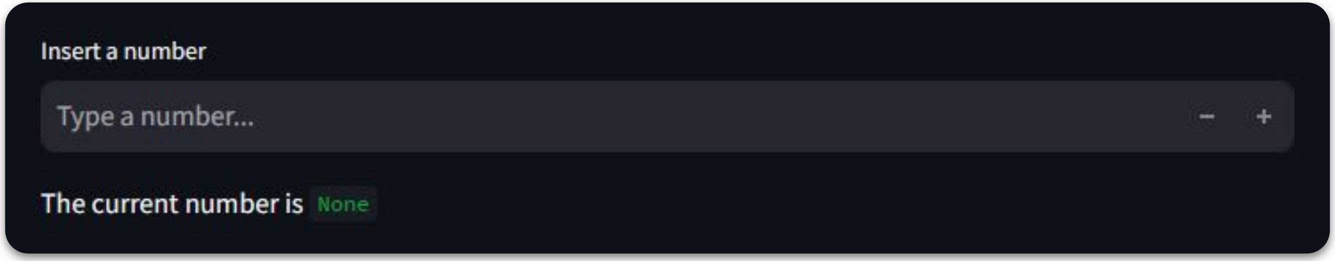
```
[
  0 : "Green"
  1 : "Red"
]
```

## Number Input

The **`st.number_input`** displays a numeric input widget



```python
import streamlit as st

number = st.number_input(
    "Insert a number", value=None, placeholder="Type a number..."
)
st.write("The current number is ", number)
```



Insert a number

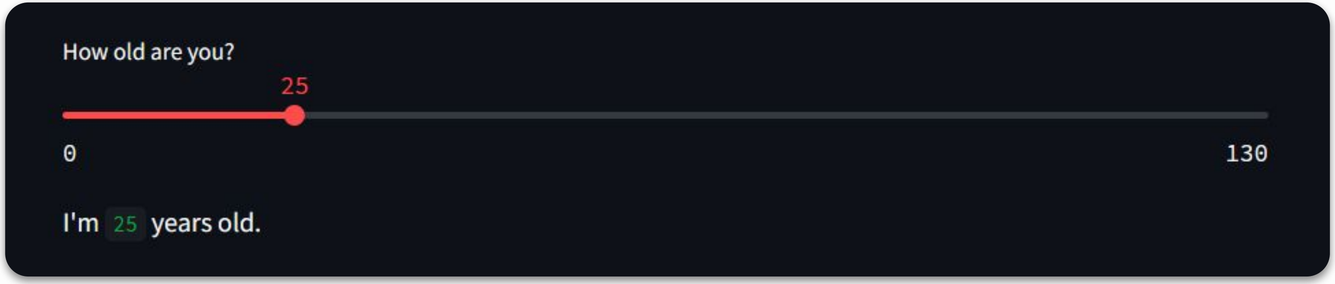
Type a number...

The current number is `None`

## Slider

The **`st.slider`** displays a slider widget for integers, time, and datetime values

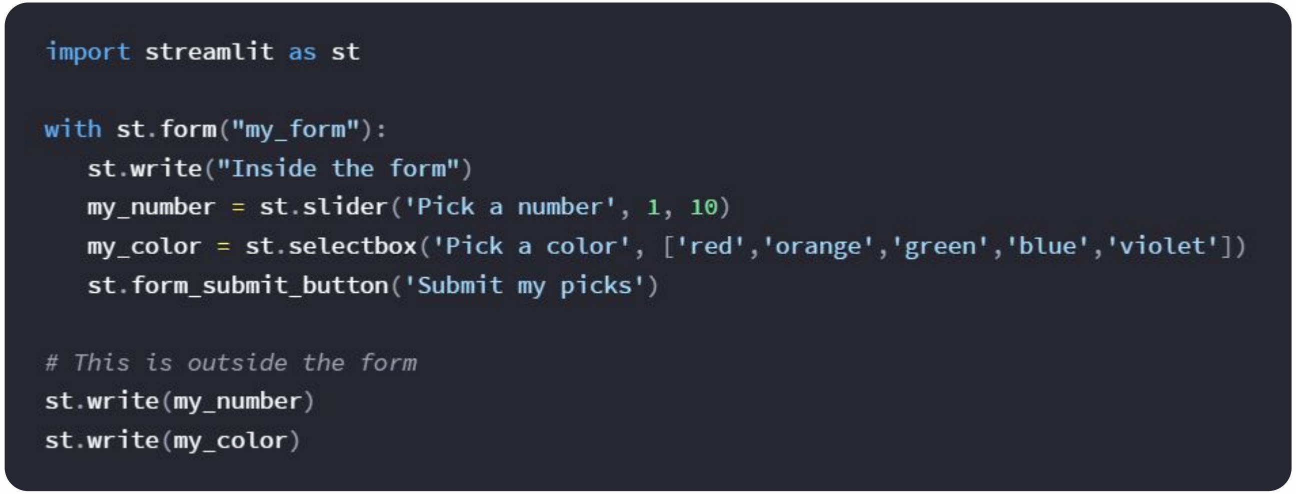
```python
import streamlit as st

age = st.slider("How old are you?", 0, 130, 25)
st.write("I'm ", age, "years old")
```



I'm 25 years old.

## Submit Form

The **`st.form`** ensures that every input change doesn’t refresh the page every time



```python
import streamlit as st

with st.form("my_form"):
   st.write("Inside the form")
   my_number = st.slider('Pick a number', 1, 10)
   my_color = st.selectbox('Pick a color', ['red','orange','green','blue','violet'])
   st.form_submit_button('Submit my picks')

# This is outside the form
st.write(my_number)
st.write(my_color)
```

## Data Handling

Process and visualize more data-intensive processes

## Upload Files

Run the following on your chosen terminal to setup commits and remote connections

```python
import streamlit as st

uploaded_files = st.file_uploader(
    "Choose a CSV file", accept_multiple_files=True
)
for uploaded_file in uploaded_files:
    bytes_data = uploaded_file.read()
    st.write("filename:", uploaded_file.name)
    st.write(bytes_data)
```

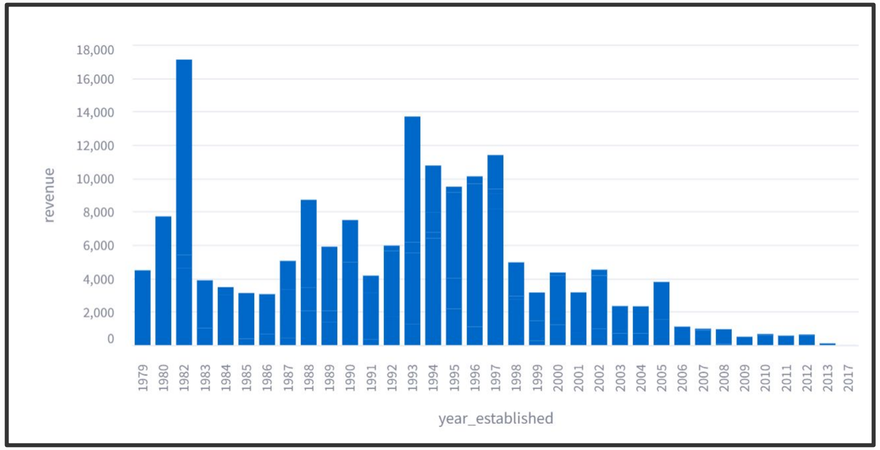
## Read CSV and Excel File

Run the following on your chosen terminal to setup commits and remote connections

```
import streamlit as st
import pandas as pd

uploaded_file = st.file_uploader("File:", type=["csv", "xlsx", "xls"])

if uploaded_file is not None:
    st.write(f"Uploaded file: {uploaded_file.name}")

    if uploaded_file.name.endswith(".csv"):
        df = pd.read_csv(uploaded_file)
    elif uploaded_file.name.endswith((".xlsx", ".xls")):
        df = pd.read_excel(uploaded_file)

    st.write(df)
```

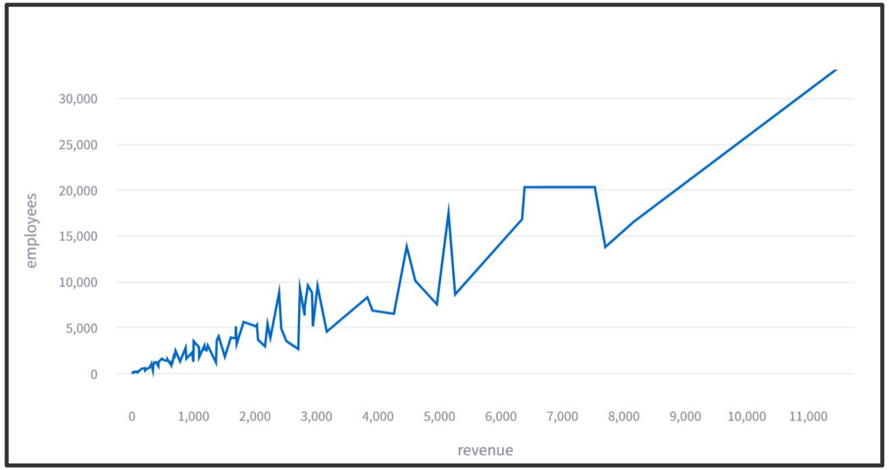
## Bar Chart

```python
import streamlit as st
import pandas as pd

df = pd.read_csv("data/sales/accounts.csv")
st.bar_chart(df, x="year_established", y="revenue")
```



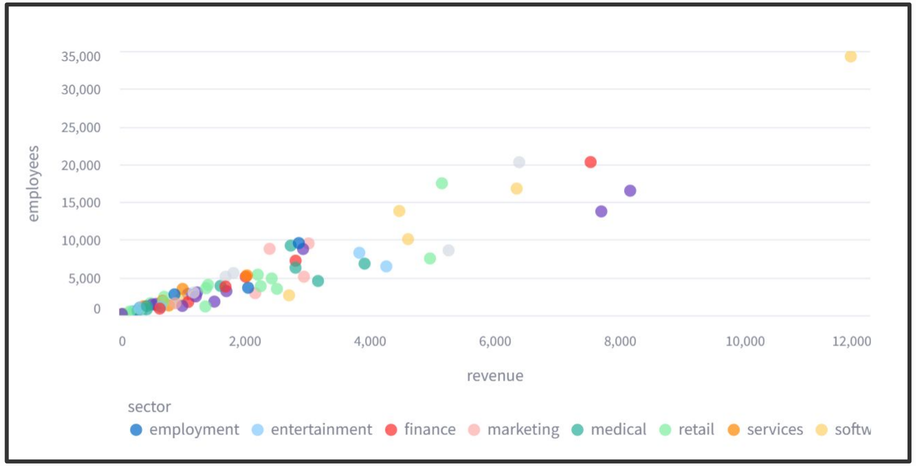
## Line Plot

```python
import streamlit as st
import pandas as pd

df = pd.read_csv("data/sales/accounts.csv")
st.line_chart(df, x="revenue", y="employees")
```



## Scatter Chart

```python
import streamlit as st
import pandas as pd

df = pd.read_csv("data/sales/accounts.csv")
st.scatter_chart(df, x="revenue", y="employees", color="sector")
```

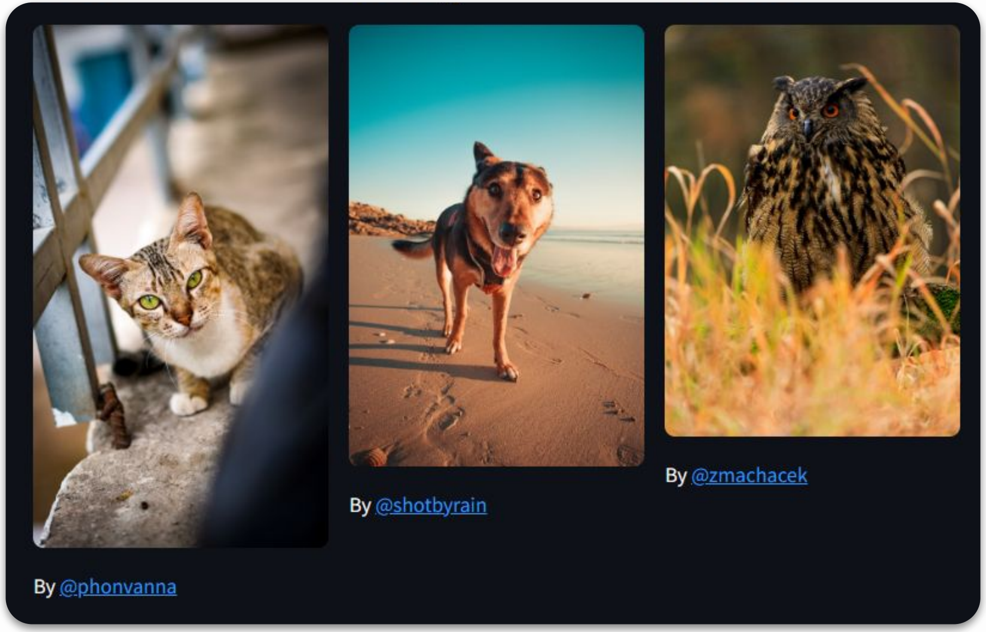


## Modularization

High-level Streamlit code organization

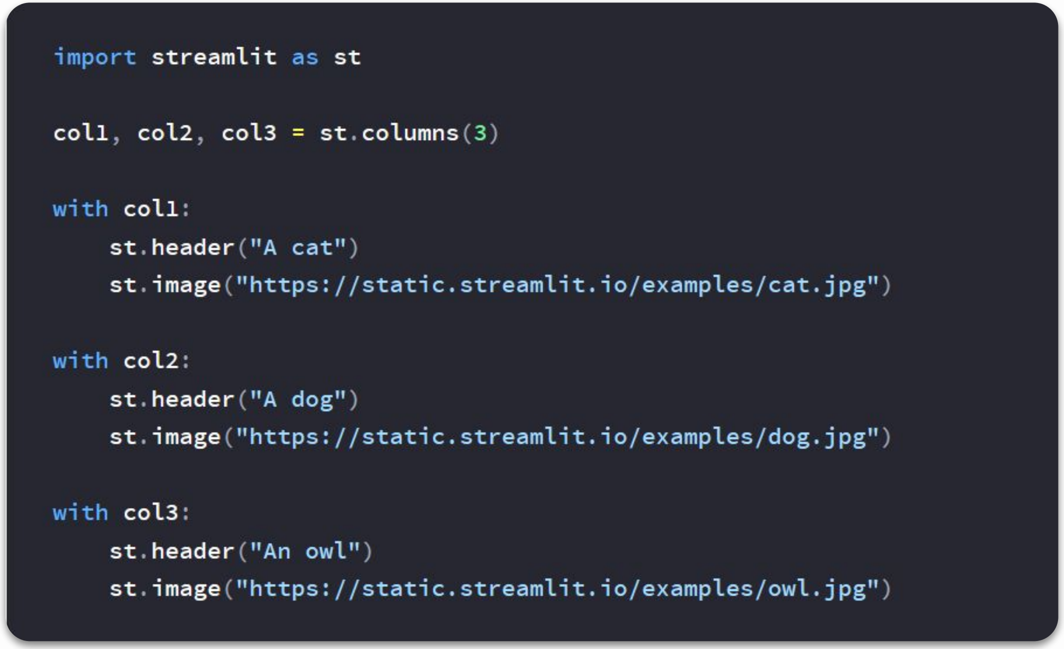
## Column Layouting

Streamlit supports multi-column layouts



## Columns

Using the context handler **`with`** syntax, content will be divided into separate columns

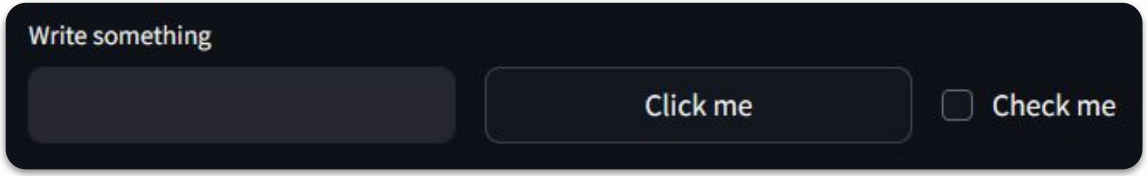


```python
import streamlit as st

col1, col2, col3 = st.columns(3)

with col1:
    st.header("A cat")
    st.image("https://static.streamlit.io/examples/cat.jpg")

with col2:
    st.header("A dog")
    st.image("https://static.streamlit.io/examples/dog.jpg")

with col3:
    st.header("An owl")
    st.image("https://static.streamlit.io/examples/owl.jpg")
```

## Simple Column Layout

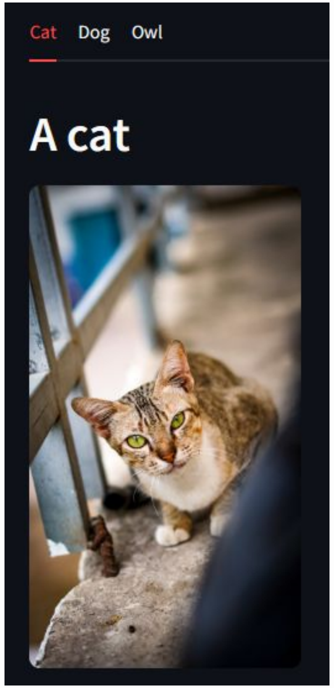
For simple columns, **`st`** can be replaced with the given column name

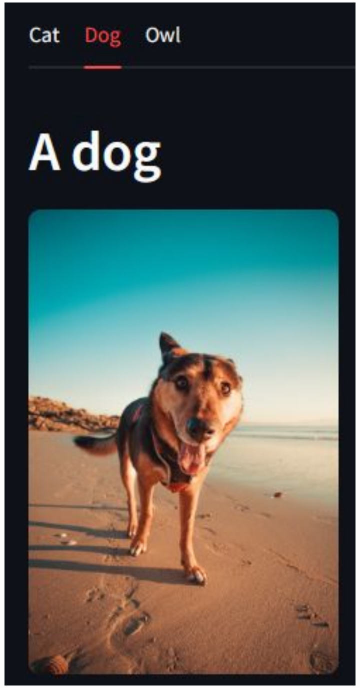
```python
import streamlit as st

left, middle, right = st.columns(3, vertical_alignment="bottom")

left.text_input("Write something")
middle.button("Click me", use_container_width=True)
right.checkbox("Check me")
```

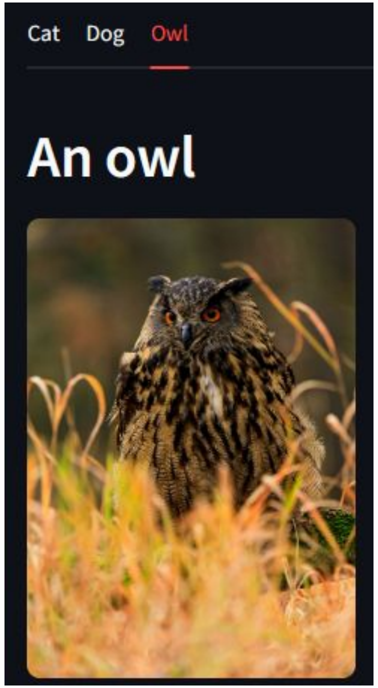


## Tabs

Streamlit also supports tab layouts to prevent cluttering the page

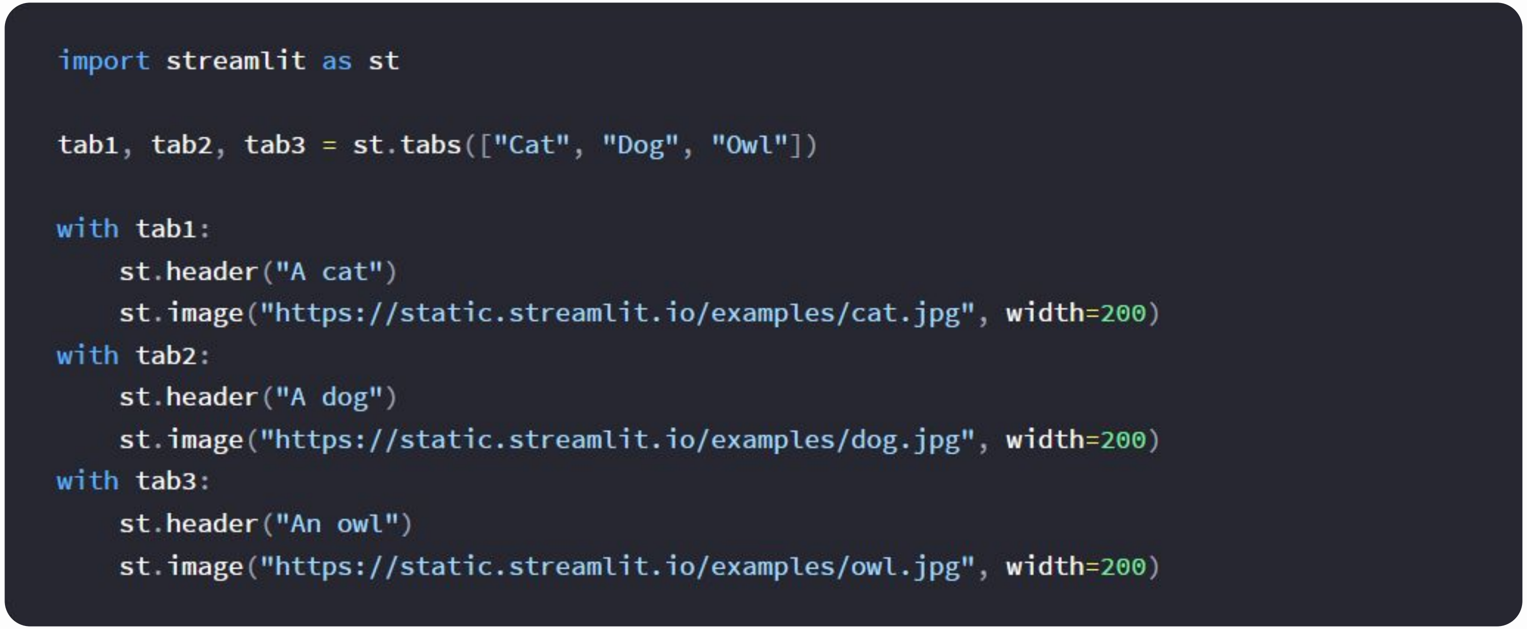






## Tabs

Using the context handler **`with`** syntax, content will be divided into separate tabs



```python
import streamlit as st

tab1, tab2, tab3 = st.tabs(["Cat", "Dog", "Owl"])

with tab1:
    st.header("A cat")
    st.image("https://static.streamlit.io/examples/cat.jpg", width=200)
with tab2:
    st.header("A dog")
    st.image("https://static.streamlit.io/examples/dog.jpg", width=200)
with tab3:
    st.header("An owl")
    st.image("https://static.streamlit.io/examples/owl.jpg", width=200)
```

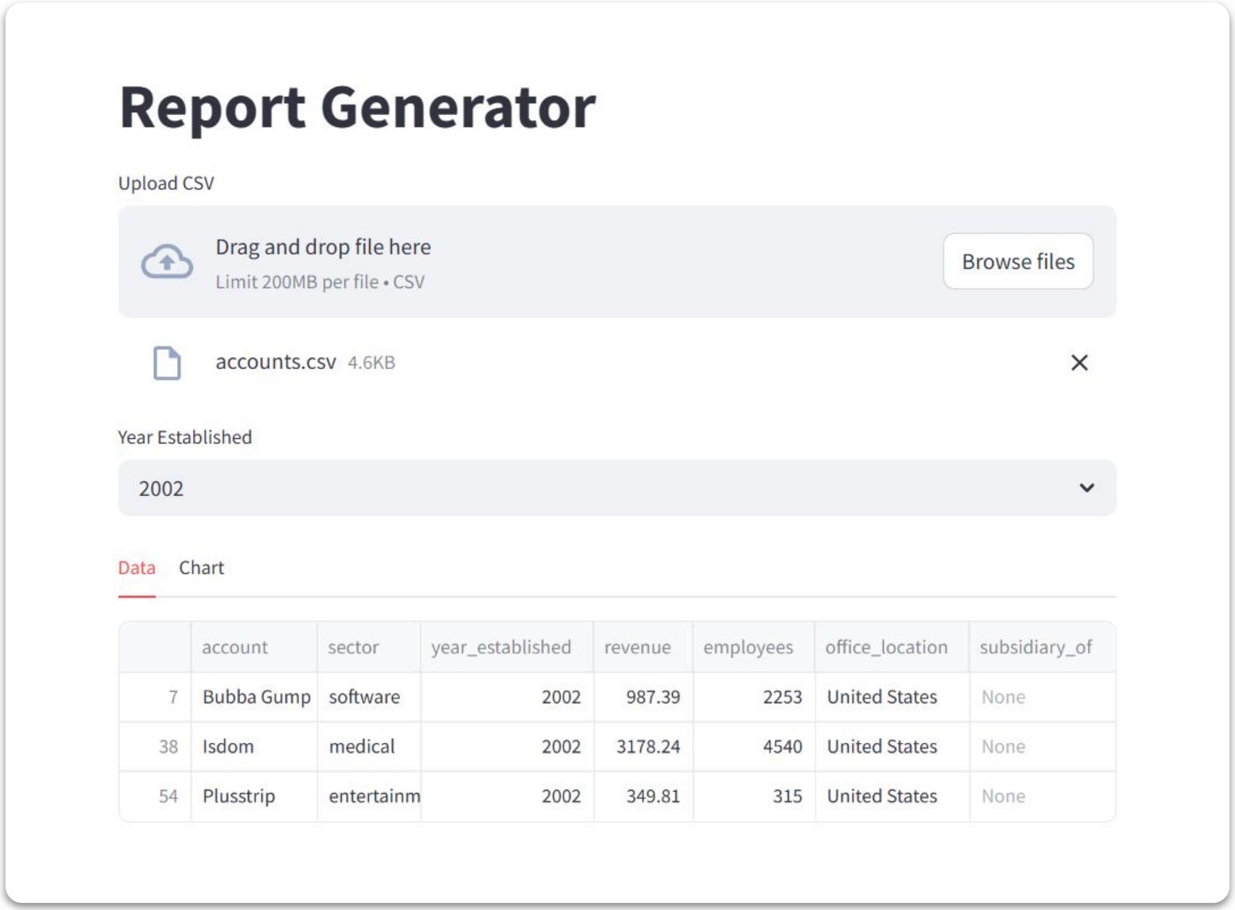
## Multiple Pages

Multiple subpages are easy to implement in Streamlit. Place subpages in the **pages/** folder

```
.
└── project_name/
    ├── ...
    └── src/
        ├── pages/
        │   ├── subpage1.py
        │   ├── subpage2.py
        │   └── subpage3.py
        └── main.py
```

## Report Generator

Visual demonstration of Streamlit’s capabilities



# Report Generator

Upload CSV

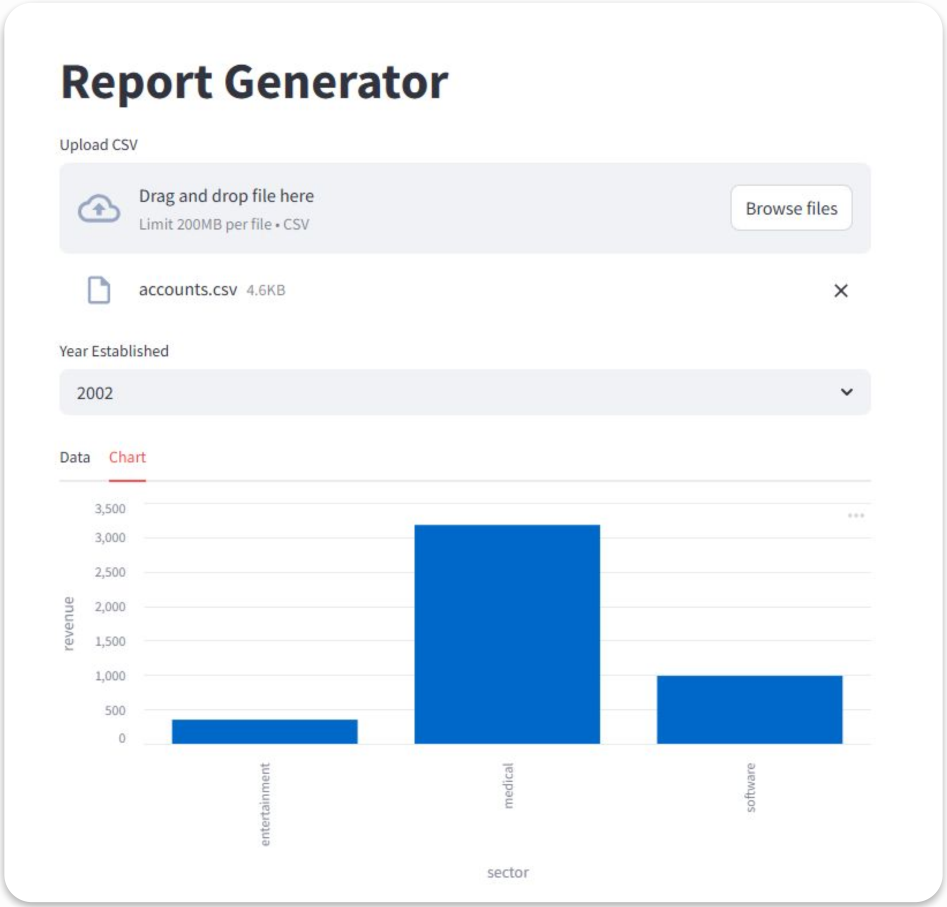
Drag and drop file here
Limit 200MB per file • CSV

accounts.csv 4.6KB

Year Established

2002

| | account | sector | year_established | revenue | employees | office_location | subsidiary_of |
|---|---|---|---|---|---|---|---|
| 7 | Bubba Gump | software | 2002 | 987.39 | 2253 | United States | None |
| 38 | Isdom | medical | 2002 | 3178.24 | 4540 | United States | None |
| 54 | Plusstrip | entertainm | 2002 | 349.81 | 315 | United States | None |



# Report Generator

Upload CSV

Drag and drop file here

Limit 200MB per file • CSV

Browse files

accounts.csv 4.6KB

Year Established

2002

Data Chart

06

## Lab Session

The culmination of all tools

# Business Sim

## Business Simulation

You will develop a simulation for a medium-sized business with the following specifications:

- Data Representation
    - Employee System: class to represent employee data (ID, name, position, salary).
    - Customer System: A class to represent customer data (ID, name, contact, order history, etc.).
    - Payroll System: A class to calculate payroll for employees (salary, bonuses, deductions, etc.)
- Simulator
    - Generate random or dummy employees and customers
    - Simulate the passage of time depending on the user’s choice
- Key Metrics: Determine important insights and connections with data handling
- Steamlit: Generate key visualizations for key metrics

# Python: Day 04

Excel Automation