Schedule module

schedule.every(10).seconds.do(job)	Every N seconds
schedule.every(5).minutes.do(job)	Every N minutes
schedule.every().hour.do(job)	Every N hours
schedule.every().day.at("10:30").do(job)	Daily at specific time
schedule.every().monday.do(job)	Weekly
schedule.every().friday.at("18:00").do(job)	Weekly at time
schedule.every(3).hours.do(job)	Custom interval
schedule.run_pending()	Run due jobs
schedule.cancel_job(job_instance)	Cancel a job
schedule.every().day.at("12:00").do(greet, name="Kush")	Schedule tasks with arguments

Types of errors

NameError	Doesn't recognize the name you are using
TypeError	When you try to combine or manipulate data in a way python doesn't allow
IndexError	The index doesn't exist
KeyError	When you try to access a value in a dictionary using a key that doesn't exist
ZeroDivisionError	When you divide a number by 0
ValueError	Function recieves a correct type but invalid value
AttributeError	Invalid attribute or method for an object
ImportError / ModuleNotFoundError	Failed to import a module
FileNotFoundError	File does not exist when trying to open it

Pandas module

df = pd.DataFrame(dictionary)	To convert a dictionary into a pandas dataframe
df = pd.read_csv('file.csv')	To convert a csv file into a dataframe
df = pd.read_excel('file.xlsx')	To convert an excel file into a dataframe
df = pd.read_json('file.json')	To convert a json file into a dataframe
df.to_csv('output.csv', index=False)	Convert a dataframe into a csv file
df.to_excel('output.excel')	Convert a dataframe into an excel file
df.head(k)	First k rows, leave empty for five
df.tail(k)	Last k rows, leave empty for five
df.info()	Data types and non-null values
df.describe()	Summary statistics
df.shape	No. of rows and columns
df.columns	Column names
df.dtypes	Data types
df['col']	A specified column
df.iloc[k, l]	A specified cell by index, leave l empty for an entire row
df.loc[k, 'col']	A specified cell by index, 'col' is column name
df[0:5]	Slicing rows
df[df['col'] > 25]	Filter data by condition
df[df['col'] > 25 & (df['Age'] < 40)]	Filter data by multiple conditions
df[df['Name'].isin(['Alice'])]	Filter by values
df.rename(columns={'old': 'new'})	Renaming a column
df.drop(columns=['Col1', 'Col2'])	Dropping columns
df.drop(index=[0, 1])	Dropping rows
df[col].sum()	Sum of values in col

Pandas module (cont)	
df[col].mean()	Mean of values in col
df[col].value_counts()	Number of values in col
df.groupby(col).mean()	Grouped stats
df.isnull()	Returns null values of boolean dataframes
df.isnull().sum()	No. of null values
df.dropna()	Drop the row with null values
df.fillna(k)	Fill the missing values with value k
df['col'] = df['col'].str.strip()	Remove whitespace
df['col'] = df['col'].str.lower()	Present data in lowercase
df['col'] = pd.to_datetime(-df['col'])	Convert to datetime
df.sort_values('Age')	Sort data by age
df.sort_values(['Age', 'Name'])	Sort data by multiple values
df.reset_index(drop=True)	Reset index
pd.concat([df1, df2])	Appending rows
pd.merge(df1, df2, on='ID')	Joining data by column value
pd.merge(df1, df2, how='left', on='ID')	Left joining data by column value
df.pivot_table(index='Gender', values='Age', aggfunc='mean')	Create a pivot table with mean of the values categorized by index

MIME module	
msg = MIMEText('This is a plain text email body', 'plain')	Define message in plain format
msg['Subject'] = 'Plain Text Email'	Define subject
msg['From'] = 'sender@example.com'	Define sender's address
msg['To'] = 'recipient@example.com'	Define recipient's address
msg.attach(content)	Attaching a content
msg = MIMEMultipart('alternative')	Creating both versions of text

Matplotlib module	
plt.plot(x, y, color='red', linestyle='--', marker='o', label='line 1')	Line plot with color red, dashed lines, o marker labeled as 'line 1'
plt.title("Title")	Set title of the chart
plt.xlabel("x-axis")	Label of x-axis
plt.ylabel("y-axis")	Label of y-axis
plt.legend()	Show legend
plt.grid(True)	Show grid
plt.show()	Display the chart
plt.figure(figsize=(6, 4))	Set figure size
plt.subplot(2, 1, 1)	2 rows, 1 column, 1st plot
plt.tight_layout()	Avoid overlap
plt.scatter(x, y)	Scatter plot
plt.bar(x, y)	Bar plot
plt.barh(x, y)	Horizontal bar plot
plt.hist(list, bins=5)	Histogram plot
plt.pie(data_list, labels=label_list, autopct='%1.1f%%')	Pie chart plot
plt.style.use('ggplot')	Set global chart style
plt.style.available	Show all chart styles
plt.savefig('plot.pdf', dpi=300)	Save chart as pdf with resolution
plt.savefig('plot.png')	Save chart as png
plt.text(2, 20, "Sample Text")	Add sample text to x=2, y=20
plt.annotate("Important", xy=(2, 20), xytext=(3, 25), arrowprops=dict(facecolor='black'))	For annotating
plt.xscale('log')	Logarithmic x-axis
plt.yscale('log')	Logarithmic y-axis
plt.xlim(0, 5)	X-axis limits
plt.ylim(0, 5)	Y-axis limits
plt.xticks([1, 2, 3])	Custom ticks in x-axis
plt.yticks([1, 2, 3])	Custom ticks in y-axis

Requests module

response = requests.get('https://api.example.com/data')	GET request
response = requests.post('https://api.example.com/create', data={'key': 'value'})	POST request
response = requests.put('https://api.example.com/update/1', data={'key': 'new_value'})	PUT request
response = requests.delete('https://api.example.com/delete/1')	DELETE request
response.status_code	Status Code
response.headers	headers dictionary
response.text	Raw response as text
response.json()	Parse response as JSON
requests.get('https://example.com', proxies=proxies)	Request with proxy

Plotly module

import plotly.graph_objects as go	
import plotly.express as px	
df = px.data.gapminder()	Returning a Gapminder dataset as a pandas dataframe
px.line(df[df['country'] == 'India'], x='year', y='gdpPercap', title='GDP over time')	Line plot country dataframe, x=year, y=gdppercap and title is GDP over time
px.bar(x=['A', 'B'], y=[10, 20], title='Bar Plot')	Bar plot
px.scatter(df, x='gdpPercap', y='lifeExp', color='continent', title='GDP vs Life Expectancy')	Scatter plot
px.scatter(df, x='gdpPercap', y='lifeExp', size='pop', color='continent', hover_name='country', log_x=True)	Bubble sort

Plotly module (cont)

px.choropleth(df[df['year']==2007], locations="iso_alpha", color="lifeExp", hover_name="country")	Map plot (Choropleth)
fig.update_layout(title='New Title', xaxis_title='X Axis', yaxis_title='Y Axis', template='plotly_dark')	To customize layout
fig.add_trace(go.Scatter(x=[1, 2, 3], y=[4, 5, 6], mode='lines+markers', name='Line'))	Line plot
fig = go.Figure(go.Bar(x=['A', 'B'], y=[10, 15]))	Bar plot
go.Figure(go.Pie(labels=['A', 'B'], values=[30, 70]))	Pie plot
fig.write_html("plot.html")	Save as html file
fig.write_image("plot.png")	Save as image file
fig.update_layout(hovermode='x unified')	Tooltip follows x
fig.update_traces(marker=dict(size=10))	Change marker size
fig.update_layout(dragmode='zoom')	Default zoom tool
fig.update_layout(template='plotly_dark')	Update the style of theme
px.scatter_geo(px.data.gapminder().query("year==2007"), locations="iso_alpha", color="continent", size="pop")	Map visualizations
from plotly.subplots import make_subplots	
fig = make_subplots(rows=1, cols=2)	To set subplots
fig.add_trace(go.Scatter(x=[1, 2], y=[3, 4]), row=1, col=1)	add trace in a subplot

Random module

random.random()	random float between 0.0 and 1.0
random.uniform(a, b)	random float between a and b
random.randint(a, b)	random integer between a and b
random.randrange(0, 10, 2)	random number from [0, 2, 4, 6, 8, 10]
random.choice(list)	random element from a list
random.choices(list, weights=None, k=2)	k no. of random elements from a list with replacement, weights is a list that specifies the probability of choosing a specific element
random.sample(list, k=2)	k no. of unique elements(no replacement)
random.shuffle(list)	shuffles a list
random.seed(a=None)	use this to get the same result every time

SMTPlib module

server = smtplib.SMTP('smtp.gmail.com', 587)	Connect with SMTP server through TLS
server.starttls()	Start TLS connection
server = smtplib.SMTP_SSL('smtp.gmail.com', 465)	Connect with SMTP server through SSL
server.login('your_email@example.com', 'your_password')	Login to your account
server.sendmail(from_email, to_email, message)	To send mail from your email
server.quit()	Close the connection (Very Important)

Glob module

glob.glob('*.txt')	All .txt files in current directory
glob.glob('*/*.txt', recursive=True)	Match files in subdirectories
glob.glob('*.txt', recursive=False, include_hidden=False)	Sort matched files

Glob module works best when worked with Regex expressions

Os module

os.getcwd()	Returns the current working directory
os.chdir('path/to/directory')	Changes current working directory
os.listdir('path')	Lists files and folders in the specified path
os.mkdir('dirname')	Creates a single directory
os.makedirs('dir/subdir')	Creates intermediate directories as needed
os.rmdir('dirname')	Removes an empty directory
os.removedirs('dir/subdir')	Removes nested empty directories
os.remove('filename')	Removes a file
os.path.exists('path')	Returns true if path exists
os.path.isfile('path')	True if it's a file
os.path.isdir('path')	True if it's a directory
os.path.join('folder', 'file.txt')	Combines path using the right seperator
os.path.basename('path/to/file.txt')	Returns file.txt
os.path.dirname('path/to/file.txt')	Returns 'path/to'
os.path.split('path/to/file.txt')	Returns ('path/to', 'file.txt')
os.path.abspath('file.txt')	Returns absolute path to the file
os.environ.get('HOME')	Retrieves the path to the current user's home directory
os.environ['MY_VAR'] = 'value'	Sets or creates an environment variable within the current environment process
os.system('ls')	Executes a shell command
os.getpid()	Current process ID
os.getppid()	Current parent process ID
os.rename('old.txt', 'new.txt')	Renaming a file or a directory

IMAPlib module

imap = imaplib.IMAP4_SSL('imap.gmail.com', 993)	Connect to mail server
imap.login('your_email@example.com', 'your_password')	Login to your account
imap.list()	List all messages
imap.select('INBOX')	Select a mailbox
status, messages = imap.search(None, 'ALL')	Search all emails
status, messages = imap.search(None, 'UNSEEN')	Search unread emails
status, messages = imap.search(None, 'FROM', '"sender@example.com"')	Search emails from a specific address
imap.store(latest_email_id, '+FLAGS', '\\Seen')	Mark emails as read
imap.store(latest_email_id, '+FLAGS', '\\Deleted')	Delete all emails
imap.close(), imap.logout()	Close connection (Very important)

Re module

re.search(pattern, string)	Searches for first match everywhere
re.match(pattern, string)	Checks for a match only at the beginning
re.fullmatch(pattern, str)	Matches entire string to pattern
re.findall(pattern, string)	Returns all non-overlapping matches
re.finditer(pattern, string)	Returns iterators yielding match objects
re.sub(pat, repl, string)	Replace matches with repl
re.split(pattern, string)	Split string by the matches
re.compile(pattern)	Precompile a pattern for reuse
Regex Expressions used in Python:	
.	Matches any character except newline, use like this: a.b.c to match a1b7c, asbfc, a9bkc etc.

Re module (cont)

?	Use after a character to define it occurs 0 or 1 times
\	To define a Regex pattern / Escape character
*	Use after a pattern to define 0 or more repetitions
+	Use after a pattern to define 1 or more repetitions
^	Use before a pattern to define start of a string
$	Use after a pattern to define end of string
{k}	Use to define k number of repetitions for a pattern
{k, l}	Use to define between k and l repetitions
[]	define a list of characters and use if you match from one of them
\d	Specifies digits [0-9]
\D	Anything that's not a digit
\w	Any word character [a-zA-Z0-9_]
\W	Anything that is not a word character
\s	Whitespace \ Spacing between two words
\S	Non-whitespace
\b	Word boundary, used to match whole words only like: \bcat\b to match 'cat', 'little cat' and not 'tomocat' or 'catatine'
\B	Non-word boundary, best used to match a word which has that letter like: \bun\b matches 'unmalicious', 'unnasty' and not 'un' or 'we un'

Shutil module

shutil.copy(src, destination)	Copies file to destination
shutil.copy2(src, dst)	It's like copying but preserves metadata
shutil.copymode(src, dst)	Copies file permissions only
shutil.copystat(src, dst)	Copies file's metadata only
shutil.copytree(src, dst	Copies entire directory tree
shutil.move(src, dst)	Moves or renames a file
shutil.rmtree('dir')	Deletes directory and everything in it
shutil.make_archive(base_name, format, root_dir)	Creates archive in any format
shutil.unpack_archive(filename, extract_dir)	Unpacks the archive
shutil.disk_usage(path)	Gets disk usage stats

Bokeh module

from bokeh.plotting import figure, show	
from bokeh.io import output_file, output_notebook	
from bokeh.layouts import column, row	
output_file("plot.html")	Output to html file
output_notebook()	Output to Jupyter notebook
p = figure(title="Simple Line", x_axis_label='x', y_axis_label='y')	Label the figure
p.line([1, 2, 3], [4, 6, 2])	Line plot
show(p)	Show the chart
p.circle(x, y, size=10)	Scatter plot
p.vbar(x=x, top=y, width=0.5)	Vertical bar plot
p.hbar(x=x, top=y, width=0.5)	Horizontal bar plot
p.triangle(x, y, size=12, color="-green")	Shape plot, other glyphs available ex: square, diamond etc.
p.title.text = "Custom Title"	Set title
p.xaxis.axis_label = "X Axis"	Label x-axis
p.yaxis.axis_label = "Y Axis"	Label y-axis

Bokeh module (cont)

p.background_fill_color = "lightgray"	Set background color
p.border_fill_color = "whitesmoke"	Set border color
p.outline_line_color = "black"	Set outline line color
p.line(x, y, legend_label="My Line", line_width=2)	define legend_label for legend
p.legend.location = "top_left"	Set interactive legend
p.legend.click_policy = "hide"	
layout = row(p1, p2)	To set layout of a row
layout = column(p1, p2)	To set layout of a column
show(layout)	Show layout
from bokeh.models import ColumnDataSource	
source = ColumnDataSource(data={'x': [1, 2, 3], 'y': [4, 6, 5]})	Set a data source
p.circle(x='x', y='y', source=source, size=10)	Plot a circle chart from data source
from bokeh.io.export import export_png	
export_png(p, filename="plot.png")	Export chart to png file
p1.x_range = p2.x_range	Link x-axis
p1.y_range = p2.y_range	Link y-axis
from bokeh.embed import components	
script, div = components(p)	Use in html templates

Numpy module

np.array([1, 2, 3], [4, 5, 6])	Creating a 2D array
np.zeros((3, 3))	3x3 array of zeros
np.ones((3, 3))	3x3 array of ones
np.full((2, 2), 7)	2x2 array of sevens
np.eye(3)	Identity matrix 3x3
np.arrange(0, 10, 2)	An array of this: [0, 2, 4, 6, 8]
np.linspace(0, 1, 5)	5 values from 0 to 1
arr.shape	Dimensions of the array
arr.ndim	No. of dimensions
arr.size	Total no. of elements

Numpy module (cont)	
arr.dtype	Data type
arr.reshape((2, 3))	Reshape an array to 2x3
arr.ravel()	Compress an array to 1D
arr.T	Transpose the array
np.add(a, b)	a + b
np.subtract(a, b)	a - b
np.multiply(a, b)	a * b
np.divide(a, b)	a / b
np.power(a, 2)	a to the power of 2
np.sqrt(a)	Square root of a
np.exp(a)	Exponential value of a
np.log(a)	Natural log of a
np.mean(list)	Mean of the list
np.median(list)	Median of the list
np.std(list)	Standard deviation of the list
np.sum(list)	Sum of the list
np.max(list)	Maximum value in a list
np.min(list)	Minimum value in a list
np.argmax(list)	Index of maximum value
np.argmin(list)	Index of minimum value
np.concatenate([a, b])	Join arrays
np.vstack([a, b])	Stack vertically
np.hstack([a, b])	Stack horizontally
np.split(a, 3)	Split the array into 3 parts
np.unique(a)	Unique elements of the array
np.random.rand(2, 2)	a 2x2 array of random elements from 0 to 1
np.random.randn(2, 2)	a 2x2 array of random elements, this will be a normal distribution
np.random.randint(0, 10, size=5)	a 1D array of 5 random integers from 0 to 10
np.isnan(a)	Check for NaN values
np.isinf(a)	Check for Inf values
np.nan_to_num(a)	Convert NaN to 0
np.clip(a, 0, 1)	Limit values between 0 to 1
np.where(a > 0, 1, 0)	Conditional values
np.cumsum(a)	Cumulative sum
np.cumprod(a)	Cumulative product

Pytest module	
assert result == k	checks if the result variable is the same as the variable assigned as k
@pytest.fixture	to define a fixture to use as a reusable piece of code to use before or after a test
@pytest.mark.parametrize("a, b, result", [(1, 2, 3), (4, 5, 9)])	checks the result variable with a and b by performing numerous tests based on the data we give
@pytest.mark.skip(reason="Not implemented yet")	skip a particular test
@pytest.mark.skipif(condition, reason="...")	skip the test given the condition
@pytest.mark.xfail	If you are expecting a test to fail
pytest.raises()	to raise a specific type of error

Types of data structures	
Lists	Indexing, Slicing, Extending and Mutability, syntax: my_list = [1, 1.21, "hello", True]
Tuples	Indexing, Slicing and Immutable, syntax: my_tuple = (1, 10, "hello")
Sets	Unordered nature, Key operations are add(), remove(), union(), intersection(), difference(), syntax: my_set = {1, 2, 3, 3}
Dictionary	Accessing values by key, Mutability and flexibility, common operations are get(), items(), keys(), values(), update(), syntax: my_dict = {"name": "Alice", "age": 30, "city": "New York"}