

Schedule module

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

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

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

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

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



Requests module

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

Plotly module

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

Plotly module (cont)

<code>px.choropleth(df[df['year']==2007], locations="iso_alpha", color="lifeExp", hover_name="country")</code>	Map plot (Choropleth)
<code>fig.update_layout(title='New Title', xaxis_title='X Axis', yaxis_title='Y Axis', template='plotly_dark')</code>	To customize layout
<code>fig.add_trace(go.Scatter(x=[1, 2, 3], y=[4, 5, 6], mode='lines+markers', name='Line'))</code>	Line plot
<code>fig = go.Figure(go.Bar(x=['A', 'B'], y=[10, 15]))</code>	Bar plot
<code>go.Figure(go.Pie(labels=['A', 'B'], values=[30, 70]))</code>	Pie plot
<code>fig.write_html("plot.html")</code>	Save as html file
<code>fig.write_image("plot.png")</code>	Save as image file
<code>fig.update_layout(hovermode='x unified')</code>	Tooltip follows x
<code>fig.update_traces(marker=dict(size=10))</code>	Change marker size
<code>fig.update_layout(dragmode='zoom')</code>	Default zoom tool
<code>fig.update_layout(template='plotly_dark')</code>	Update the style of theme
<code>px.scatter_geo(px.data.gapminder().query("year==2-007"), locations="iso_alpha", color="continent", size="pop")</code>	Map visualizations
<code>from plotly.subplots import make_subplots</code>	
<code>fig = make_subplots(rows=1, cols=2)</code>	To set subplots
<code>fig.add_trace(go.Scatter(x=[1, 2], y=[3, 4]), row=1, col=1)</code>	add trace in a subplot

Random module

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

SMTPlib module

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

Glob module

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

Glob module works best when worked with Regex expressions

Os module

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

IMAPlib module

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

Re module

<code>re.search(pattern, string)</code>	Searches for first match everywhere
<code>re.match(pattern, string)</code>	Checks for a match only at the beginning
<code>re.fullmatch(pattern, str)</code>	Matches entire string to pattern
<code>re.findall(pattern, string)</code>	Returns all non-overlapping matches
<code>re.finditer(pattern, string)</code>	Returns iterators yielding match objects
<code>re.sub(pattern, repl, string)</code>	Replace matches with repl
<code>re.split(pattern, string)</code>	Split string by the matches
<code>re.compile(pattern)</code>	Precompile a pattern for reuse

Regex Expressions used in Python:

<code>.</code>	Matches any character except newline, use like this: <code>a.b.c</code> to match <code>a1b7c</code> , <code>asbfc</code> , <code>a9bkc</code> etc.
----------------	--

Re module (cont)

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



Shutil module

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

Bokeh module

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

Bokeh module (cont)

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

Numpy module

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

Numpy module (cont)

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

Pytest module

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

Types of data structures

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