

## ✓ Function as Argument

Function is a block of code that executes when it is called.

It is also known as object that can be passed to the another functions.

It can be used across the whole program.

---

To create a Python function, use the **def** keyword followed by a valid function name, following variable naming rules.

## ✓ Simple Fucntion

```
1 # @title Simple Fucntion
2 # creating a function
3
4 def addition(n): # function with one variable argument
5     return n + n # return the function result
6
7 num = 6 # input variable
8 x = addition(num) # calling the function & store in one variable to view the result
9
10 print(x)

1 def add(n): # first function
2     return n + 2
3
4 def square(add_func): # second function with another 'function's' return as a argument
5
6     return add_func * 2 # here we are multiplying 2 with the return value of func add()
7
8 num = 3
9 y = square(add(num)) # calling the function with passing the argument as another function
10
11 print(y)
```

Click here to view the execution of the above code

[https://drive.google.com/file/d/1CZvVj\\_iClbY9R8d3CKkYJ8Ff-8GERITL/view?usp=sharing](https://drive.google.com/file/d/1CZvVj_iClbY9R8d3CKkYJ8Ff-8GERITL/view?usp=sharing)

## ✓ List Comprehension

1. It is shortcut for creating lists.
  2. Simplifying **multiple lines** of code used to append elements to a list into a **single line** is known as '**list comprehension**'.
- 

### Basic Syntax

lis = [expression for item in iterable if condition]

- 'expression' is the operation you want to perform on each item.
- 'item' represents each element in the iterable.
- 'iterable' is the source data from which you want to create the new list.
- 'condition' (optional) is a filter that determines whether an item should be included in the new list.

```
1 # example for list comprehension
2
3 n = [2, 4, 6, 8, 10]
4
5 sq = [x**2 for x in n] # iterate the element in 'n' then square it and append it to the square list
6
7 print(sq)
```

Click this link to view the execution

<https://drive.google.com/file/d/1QIWZ7PX0vcvLomrUxouLO6akvFGLBTj0/view?usp=sharing>

## ✓ File Handling

It is a fundamental concept in python that allows you to work with files, such as

1. Opening a file
2. Reading a file
3. Writing a file
4. Closing a file

## ✓ Modes

- "r" - Read (default mode)
- "w" - Write (create a new file or overwrites an existing one)
- "a" - Append (adds content to existing file at the end)
- "x" - Create (create a new file but raises an error if it already exists)

## ✓ Opening a File

```
1 # @title Opening a File
2
3 # Open a file and view it
4
5 f = open('file1.txt', 'r') # open('file_name', 'mode')
6
7 print(f) # file open and store in the variable f
8
9 view = f.read() # to view the file content
10 print(view)
11
12 f.close() # close the opened file
```

<\_io.TextIOWrapper name='file1.txt' mode='r' encoding='UTF-8'>  
Hi, Everyone!  
Here you are going to learn about the Python topics, such as

## ✓ Reading a File

```
1 # @title Reading a File
2
3 # first open the file
4
5 f = open('file1.txt', 'r')
6
7 # read entire file
8
9 v = f.read()
10 print("Entire file content:", v)
11
12 f.close()
```

Entire file content: Hi, Everyone!  
Here you are going to learn about the Python topics, such as

```
1 # read one line at a time - readline()
2
3 f = open("file1.txt", 'r')
4
5 l = f.readline()
6 print("Read one line in a file:", l)
7 f.close()
```

Read one line in a file: Hi, Everyone!

```
1 # read all lines into a list - readlines()
2
3 f = open("file1.txt", "r")
4
5 vl = f.readlines()
6 print('Read line in list:', vl)
7
8 f.close()
```

```
Read line in list: ['Hi, Everyone!\n', 'Here you are going to learn about the Python topics, such as']
```

## ✓ Writing in a Existing File

```
1 # @title Writing in a Existing File
2
3 o = open("file1.txt", "a") # mode = a(append)
4 print("The file is opened in Append mode: ", o)
5 o.close()
```

```
The file is opened in Append mode: <_io.TextIOWrapper name='file1.txt' mode='a' encoding='UTF-8'>
```

```
1 # write a line in existing file - write()
2
3 o = open("file1.txt", "a") # mode = a
4
5 w = o.write("\n1. Function as Arguments")
6 print(w) # print no of character we write
7
8 o.close()
```

```
25
```

```
1 # now check append file
2
3 a = open("file1.txt", "r")
4
5 print(a.read())
```

```
Hi, Everyone!
Here you are going to learn about the Python topics, such as
1. Function as Arguments
```

```
1 # Writind multiple lines which means append - writelines()
2
3 m = open("file1.txt", 'a')
4 x = m.writelines(["\n2. List Comprehension\n", "3. File Handling\n", "4. Lambda Function\n"])
5 print(m) # print the no. of characters
6
7 m.close()
8
```

```
<_io.TextIOWrapper name='file1.txt' mode='a' encoding='UTF-8'>
```

```
1 # now view the appending list in the file
2
3 m = open("file1.txt", 'r')
4 print(m.read())
5
6 m.close()
```

```
Hi, Everyone!
Here you are going to learn about the Python topics, such as
1. Function as Arguments
2. List Comprehension
3. File Handling
4. Lambda Function
```

## ✓ Creating New File - mode = x

```
1 # @title Creating New File - mode = x
2
3 n = open("new_file.txt", "x") # creating a new file
4
5 nf = n.write("This is a new file for your reference.") # writing content into the creating file
6 print("File created Successfully.")
7
8 n.close()
9
10 # reading a file
11 r = open("new_file.txt", 'r')
12 print("created file: ", r.read())
13 r.close()
```

```
File created Successfully.
created file: This is a new file for your reference.
```

- ✓ The recommended way to work with file is 'with' statement.

It automatically close the file once it open. Below we see the syntax,

```
1 with open("file1.txt", "r") as file:
2     content = file.read()
3     print(content)

Hi, Everyone!
Here you are going to learn about the Python topics, such as
1. Function as Arguments
2. List Comprehension
3. File Handling
4. Lambda Function
```

- ✓ 'r+' - this + operation is used to write and read file at the same time

```
1 with open("new_file.txt", "r+") as f:
2     print("Reading a File: ",f.read())
3
4     f.write("\nThis is the new line inserted by using the + operation.")
5

Reading a File: This is a new file for your reference.
This is the new line inserted by using the + operation.
```

## ✓ Lambda Function

It is small, anonymous function, and used when you need a quick function for a short period of time. (anonymous function - unnamed function)

- small - It is typically used for simple operations and are not meant for complex task.
- anonymous - Lambda function don't have a name like regular functions defined with **def**. Instead, you define them on the spot.
- short-term use - It is useful when you need a function temporarily, such as for sorting or filtering a list.

**Note:** It is single expression function.

**Syntax:** lambda arguments: expression

## ✓ Example for Lambda Function

```
1 # @title Example for Lambda Function
2
3 # creating single expression by using lambda function
4 cube = lambda v: v ** 3
5
6 # pass argument to the function
7 c = cube(6)
8
9 print(c)

216
```

Click here to check the execution of lambda function

<https://drive.google.com/file/d/1QIWZ7PX0vcvLomrUxouLO6akvFGLBTj0/view?usp=sharing>

## ✓ Map

It is a function in Python is a way to apply a given function to every item in a sequence, such as a list, and get back a new sequence with the results.

## ✓ simply say about Map function

1. You have a list or another iterable(like a tuple)
2. You want to do something to every item in that list.

3. You define a function that specifies what to do to each item.
4. You use the 'map' function to apply that function to every item in the list.
5. 'map' returns a new iterable (usually another list) with the results of applying the function to each item.

**NOTE:** <function at 0x79f4235ee710>:- This element appears to be a reference to a lambda function or function object. This indicates that it's an anonymous function.

[1, 2, 3, 4, 5]:- This element is a list containing the numbers 1 through 5.

```
1 numbers = [1, 2, 3, 4, 5]
2 squared_numbers = list((lambda x: x**2, numbers))
3 print("Before using map ", squared_numbers) # output: [<function <lambda> at 0x79f4235ef250>, [1, 2, 3, 4, 5]]
4
5 squared_numbers = list(map(lambda x: x**2, numbers))
6 print("After using map ", squared_numbers)

Before using map  [<function <lambda> at 0x79f4235ee710>, [1, 2, 3, 4, 5]]
After using map  [1, 4, 9, 16, 25]
```

## ✓ Filter

It is a function in python is used to filter elements from an iterable (like a list) based on specified condition.

It creates a new iterable containing only the elements that satisfy the condition.

### ✓ some ways to explain Filter

- You have a list or another iterable with some items.
- You want to keep only the items that meet a certain condition.
- You define a function or a condition that specifies which items to keep.
- You use the '**filter**' function to apply that function or condition to each item in the iterable.
- '**filter**' returns a new iterable (usually another list) containing only the items that meet the condition.

```
1 numbers = [1, 2, 3, 4, 5, 6]
2 even_num = list((lambda x: x % 2 == 0, numbers))
3
4 print("Before using Filter: ", even_num) # output: [<function <lambda> at 0x79f42359f0a0>, [1, 2, 3, 4, 5, 6]]
5
6 even_numbers = list(filter(lambda x: x % 2 == 0, numbers))
7 print("After using Filter: ", even_numbers)
8

Before using Filter:  [<function <lambda> at 0x79f4235ee320>, [1, 2, 3, 4, 5, 6]]
After using Filter:  [2, 4, 6]
```

## ✓ Regular Expression - RegEx

A regular expression, often referred to as regex or regexp, is a powerful tool for pattern matching within text.

It's a sequence of characters that defines a search pattern. With regular expressions, you can:

- Search for specific patterns in text.
- Validate and manipulate strings based on patterns.
- Extract specific information from text.

### Using Regular Expressions in Python:

- 're.search(pattern, text)':

Searches for a pattern in the text.

- 're.match(pattern, text)':

Matches a pattern at the beginning of the text.

- `'re.findall(pattern, text)'`:

Finds all occurrences of a pattern in the text.

- `'re.sub(pattern, replacement, text)'`:

Replaces occurrences of a pattern with the specified replacement.

## ✓ Literal Characters

```
1 # @title Literal Characters
2
3 # matches the characters
4
5 import re          # import module re for regular expression
6
7 p = 'hello'
8 text = 'Hello, world! This is a hello example, hello.'
9
10 find_match = re.findall(p, text)
11 print(find_match)

['hello', 'hello']
```

## ✓ Meta Characters

Metacharacters are special characters with predefined meanings in regular expressions.

### ✓ `.(dot)` - Matches any single character except a newline

```
1 # @title .(dot) - Matches any single character except a newline
2
3 import re
4
5 pattern = "a.b" # Matches 'a', any character, and 'b'
6 text = "acb abb aXb a12b"
7
8 matches = re.findall(pattern, text)
9 print(matches) # Output: ['acb', 'aXb', 'a12b']
```

### ✓ `*(asterik)` - Matches zero or more occurrences of the preceding character.

```
1 # @title *(asterik) - Matches zero or more occurrences of the preceding character.
2
3 import re
4
5 pattern = "ab*c" # Matches 'ac', 'abc', 'abbc', etc.
6 text = "ac abc abbc abbcc"
7
8 matches = re.findall(pattern, text)
9 print(matches) # Output: ['ac', 'abc', 'abbc', 'abbcc']
10
```

### ✓ `+(plus)` - Matches one or more occurrences of the preceding character.

```
1 # @title +(plus) - Matches one or more occurrences of the preceding character.
2
3 import re
4
5 pattern = "ab+c" # Matches 'abc', 'abbc', 'abbcc', etc., but not 'ac'
6 text = "ac abc abbc abbcc"
7
8 matches = re.findall(pattern, text)
9 print(matches) # Output: ['abc', 'abbc', 'abbcc']
10
```

### ✓ `?(question mark)` - Matches zero or one occurrence of the preceding character.

```
2
3 import re
```

- ✓ | (pipe) - Alternation, used to specify alternatives.

```
1 # @title | (pipe) - Alternation, used to specify alternatives.
2
```

- Escape Character

Some characters have special meanings in regular expressions (e.g., . or \*).

To match them literally, you can use the backslash `\` as an escape character.

```
1 # Escape Character
2
```

## Character Classes

Character classes allow you to specify a set of characters to match.

- ✓ `[@#io]` - Returns a match where one of the specified characters (@, #, i, o or ) is present.

1 # @title [@\$io] - Returns a match where one of the specified characters (@, #, i, o or \$) is present.  
2

- ✓ [A-M] - Returns a match for any uppercase character alphabetically between A and M.

```
1 # @title [A-M] - Returns a match for any uppercase character alphabetically between A and M.
2
```

- ✓ `[^xyz]` - Returns a match for any character EXCEPT x, y, and z.

```

1 # @title [^xyz] - Returns a match for any character EXCEPT x, y, and z.
2
3 import re
4
5 pattern = "[^xyz]"
6 text = "The xyz is not allowed."
7
8 matches = re.findall(pattern, text)
9 print(matches) # Output: ['T', 'h', 'e', ' ', ' ', ' ', 'i', 's', ' ', 'n', 'o', 't', ' ', 'a', 'l', 'l', 'o', 'w', 'e', 'd', '.']
10

```

✓ [3456] - Returns a match where any of the specified digits (3, 4, 5, or 6) are present.

```

1 # @title [3456] - Returns a match where any of the specified digits (3, 4, 5, or 6) are present.
2
3 import re
4
5 pattern = "[3456]"
6 text = "The numbers are 3, 4, 5, and 6."
7
8 matches = re.findall(pattern, text)
9 print(matches) # Output: ['3', '4', '5', '6']
10

```

✓ [7-9] - Returns a match for any digit between 7 and 9.

```

1 # @title [7-9] - Returns a match for any digit between 7 and 9.
2
3 import re
4
5 pattern = "[7-9]"
6 text = "The numbers are 7, 8, and 9."
7
8 matches = re.findall(pattern, text)
9 print(matches) # Output: ['7', '8', '9']
10

```

['7', '8', '9']

✓ [01][0-9] - Returns a match for any two-digit numbers from 00 to 19.

```

1 # @title [01][0-9] - Returns a match for any two-digit numbers from 00 to 19.
2
3 import re
4
5 pattern = "[01][0-9]"
6 text = "Valid hours are 00 to 19, 10 to 23."
7
8 matches = re.findall(pattern, text)
9 print(matches) # Output: ['00', '01', '02', '03', '04', '05', '06', '07', '08', '09', '10', '11', '12', '13', '14', '15', '16', '17']
10

```

['00', '19', '10']


✓ [a-zA-Z] - Returns a match for any character alphabetically between a and z (lowercase) OR between A and Z (uppercase).

```

1 # @title [a-zA-Z] - Returns a match for any character alphabetically between a and z (lowercase) OR between A and Z (uppercase).
2
3 import re
4
5 pattern = "[a-zA-Z]"
6 text = "The Quick Brown Fox Jumps Over the Lazy Dog"
7
8 matches = re.findall(pattern, text)
9 print(matches) # Output: ['T', 'h', 'e', 'i', 'c', 'k', 'r', 'o', 'w', 'n', 'o', 'x', 'u', 'm', 'p', 's', 'v', 'e', 'r', 't', 'h',
10

```

['T', 'h', 'e', 'Q', 'u', 'i', 'c', 'k', 'B', 'r', 'o', 'w', 'n', 'F', 'o', 'x', 'J', 'u', 'm', 'p', 's', 'O', 'v', 'e', 'r', 't',



✓ Predefined Character Classes

Regular expressions provide predefined character classes for common patterns



✓ `\d` - Matches any digit (equivalent to `[0-9]`).

```
1 # @title \d - Matches any digit (equivalent to [0-9]).
2
3 import re
4
5 pattern = r"\d"
6 text = "The year is 2023."
7
8 matches = re.findall(pattern, text)
9 print(matches) # Output: ['2', '0', '2', '3']
10
```

✓ `\w` - Matches any word character (letters, digits, or underscores).

```
1 # @title \w - Matches any word character (letters, digits, or underscores).
2
3 import re
4
5 pattern = r"\w"
6 text = "Hello123_world"
7
8 matches = re.findall(pattern, text)
9 print(matches) # Output: ['H', 'e', 'l', 'l', 'o', '1', '2', '3', '_', 'w', 'o', 'r', 'l', 'd']
10

['H', 'e', 'l', 'l', 'o', '1', '2', '3', '_', 'w', 'o', 'r', 'l', 'd']
```

✓ `\s` - Matches any whitespace character (spaces, tabs, newlines).

```
1 # @title \s - Matches any whitespace character (spaces, tabs, newlines).
2
3 import re
4
5 pattern = r"\s"
6 text = "This is a sentence\nwith\twhitespace."
7
8 matches = re.findall(pattern, text)
9 print(matches) # Output: [' ', ' ', ' ', ' ', ' ', '\n', '\t']
10

[' ', ' ', ' ', ' ', ' ', '\n', '\t']
```

✓ Quantifiers

Quantifiers specify how many times a character or group should be matched

✓ `{n}` - Matches exactly n times.

```
1 # @title {n} - Matches exactly n times.
2
3 import re
4
5 pattern = r"\d{3}" # Matches exactly three consecutive digits
6 text = "My phone number is 123-456-7890."
7
8 matches = re.findall(pattern, text)
9 print(matches) # Output: ['123', '456', '789']
10

['123', '456', '789']
```

✓ `{n,}` - Matches n or more times.

```
1 # @title {n,} - Matches n or more times.
2
3 import re
4
5 pattern = r"\w{5,}" # Matches words with 5 or more characters
6 text = "The quick brown fox jumps over the lazy dog in rains day."
7
8 matches = re.findall(pattern, text)
9 print(matches) # Output: ['quick', 'brown', 'jumps']
10
```

```
['quick', 'brown', 'jumps', 'rainys']
```

## ✓ {n,m} - Matches between n and m times.

```
1 # @title {n,m} - Matches between n and m times.
2
3 import re
4
5 pattern = r"\d{2,4}" # Matches 2 to 4 consecutive digits
6 text = "Valid numbers are 12, 345, 6789, and 12345."
7
8 matches = re.findall(pattern, text)
9 print(matches) # Output: ['12', '345', '6789', '1234']
10
```

## ✓ Anchor

Anchors are used to specify the position of a match within the text:

- ^ - Matches the start of a line.
- \$ - Matches the end of a line.

```
1 import re
2
3 pattern = r"^The" # Matches lines starting with "The"
4
5 text = """The quick brown fox
6 The lazy dog
7 They played in the park"""
8
9 matches = re.findall(pattern, text, re.MULTILINE)
10 print(matches)
11
12 # Output: ['The', 'The'] (Matches the start of lines 1 and 2)
13
```

```
['The', 'The', 'The']
```

```
1 import re
2
3 pattern = r"dog$" # Matches lines ending with "dog"
4
5 text = """The quick brown fox
6 The lazy dog
7 They played with the dog"""
8
9 matches = re.findall(pattern, text, re.MULTILINE)
10 print(matches)
11
12 # Output: ['dog', 'dog'] (Matches the end of lines 2 and 3)
13
```

```
['dog', 'dog']
```

## ✓ () - Groups characters together.

```
1 # @title () - Groups characters together.
2
3 import re
4
5 pattern = r"(\d{3})-(\d{2,4})" # Captures area code and number separately
6 text = "Call me at 123-45 or 7890-1234."
7
8 matches = re.findall(pattern, text)
9 print(matches) # Output: [('123', '45'), ('7890', '1234')]
10
```

## ✓ Iterators

An iterator in Python is like a tool that helps you look at items in a collection (like a list) one by one, without having to see the whole collection at once.

It's like turning the pages of a book to read it, instead of reading the entire book in one go.

**Iterable Functions:** Python provides several built-in functions that work with iterators, such as

zip(), enumerate(), map(), filter(), and range().

These functions make it easier to work with and manipulate iterators and iterables.

---

An iterator is an object that represents the stream of data produced by an iterable. It defines two methods:

`__iter__()`: Returns the iterator object itself.

`__next__()`: Returns the next element from the stream. If there are no more items, it raises the StopIteration exception.

### ✓ iter()

```
1 # @title iter()
2 my_list = [1, 2, 3, 4, 5]
3 my_iter = iter(my_list) # Get an iterator from the list
4
5 for item in my_iter:
6     print(item)
7
```

### ✓ zip()

```
1 # @title zip()
2
3 fruits = ["apple", "banana", "cherry"]
4 colors = ["red", "yellow", "red"]
5
6 # Using zip to combine fruits and colors
7 fruit_colors = zip(fruits, colors)
8
9 # Iterating through the pairs created by zip
10 for fruit, color in fruit_colors:
11     print(f"Fruit: {fruit}, Color: {color}")
12
```

```
<zip object at 0x79f4234096c0>
Fruit: apple, Color: red
Fruit: banana, Color: yellow
Fruit: cherry, Color: red
```

### ✓ enumerate() - it shows the index of the list elements

```
1 # @title enumerate() - it shows the index of the list elements
2
3 fruits = ["apple", "banana", "cherry"]
4
5 # Using enumerate to loop through fruits with their indices
6 for index, fruit in enumerate(fruits):
7     print(f"Index: {index}, Fruit: {fruit}")
8
```

```
Index: 0, Fruit: apple
Index: 1, Fruit: banana
Index: 2, Fruit: cherry
```

### ✓ Pickling

Pickling in Python is like putting your data into a magical jar so you can save it, transport it, or use it later.

It turns your Python objects, like lists or dictionaries, into a special format that can be stored in a file.

Later, you can open the jar and get your data back, exactly as it was.

It's like saving and retrieving your favorite toy for later play.

### ✓ Story for easy understanding

[https://drive.google.com/file/d/1Xa\\_HRSWf4kjlHwjcieUN4PuDJWajAn3c/view?usp=sharing](https://drive.google.com/file/d/1Xa_HRSWf4kjlHwjcieUN4PuDJWajAn3c/view?usp=sharing)

- `pickle.dump(obj, file)`: This function serializes the Python object obj and writes it to the specified file-like object file.

- `pickle.load(file)` : This function reads a serialized Python object from the specified file-like object `file` and returns the deserialized object.

```

1 # simple code for dump file
2
3 import pickle
4
5 # Sample data (a Python dictionary)
6 data = {'name': 'Alice', 'age': 30, 'city': 'Wonderland'}
7
8 # Serialize (pickle) the data
9 pickled_data = pickle.dumps(data)
10
11 # Print the pickled data
12 print(pickled_data)
13

```

b'\x80\x04\x950\x00\x00\x00\x00\x00\x00\x00}\x94(\x8c\x04name\x94\x8c\x05Alice\x94\x8c\x03age\x94K\x1e\x8c\x04city\x94\x8c\nWonderland'

```

1 import pickle # import pickle module
2
3 # Sample data (a Python dictionary)
4 data = {'name': 'Alice', 'age': 30, 'city': 'Wonderland'}
5
6 # serialize - converting python object into byte stream
7
8 # Serialize (pickle) the data and save it to a file
9
10 with open('data.pkl', 'wb') as file: # wb - writebyte
11     dumped_data = pickle.dump(data, file)
12
13 # Deserialize (unpickle) the data from the file, Deserialize - vice versa of Serialize
14
15 with open('data.pkl', 'rb') as file: # rb - readbyte
16     loaded_data = pickle.load(file)
17
18 print(loaded_data) # Output: {'name': 'Alice', 'age': 30, 'city': 'Wonderland'}
19

```

{'name': 'Alice', 'age': 30, 'city': 'Wonderland'}

- ✓ JSON File:

The keys are strings, and the values can be strings, numbers, booleans, arrays (lists), or nested JSON objects.

The `json` module in Python provides functions for encoding (serializing) and decoding (deserializing) JSON data.

- \* ``json.dumps(obj)``: Serialize a Python object `obj` into a JSON-formatted string.

- ✦ Encoding (Serialization)

```

1 # @title Encoding (Serialization)
2
3 # json.dumps() to convert Python data into JSON format
4
5
6 import json    # import json module
7
8 data = {'name': 'Alice', 'age': 30, 'city': 'Wonderland'}
9 json_string = json.dumps(data)
10
11 print(json_string)

```

## ✓ Decoding (Deserialization)

```

1 # @title Decoding (Deserialization)
2
3 # json.loads() to convert a JSON string back into a Python object
4
5 import json
6
7 json_string = '{"name": "Alice", "age": 30, "city": "Wonderland"}'
8 data = json.loads(json_string)
9
10 print(data)

```

## ✓ Encoding & Decoding

```

1 # @title Encoding & Decoding
2
3 import json
4
5 # Encoding (Serializing) Python data to JSON
6 fruits = ["apple", "banana", "cherry", "date"]
7 json_string = json.dumps(fruits)
8
9 # Decoding (Deserializing) JSON data to Python
10 decoded_fruits = json.loads(json_string)
11
12 print(decoded_fruits) # Output: ['apple', 'banana', 'cherry', 'date']

```

## ✓ Date and Time

With this module we can work with date and time. Let see the functions and operations

### ✓ Current Date & Time

```

1 # @title Current Date & Time
2
3 # fisrt import the datetime module, we mostly use the datetime class in datetime module
4
5 from datetime import datetime
6
7 current_d_t = datetime.now()
8 print(current_d_t)

```

2023-09-02 22:13:21.171554

### ✓ Date & Time Format

```

1 # @title Date & Time Format
2
3 from datetime import datetime
4
5 current = datetime.now()
6
7 format1 = current.strftime("%Y-%m-%d %H:%M:%S") # format function - strftime()
8 print("Format 1: ", format1)
9
10 format2 = current.strftime("%m-%d-%y %H:%M:%S")
11 print("Format 2: ", format2)
12
13 format3 = current.strftime("%y-%d-%m %H:%M:%S")
14 print("Format 3: ", format3)
15
16 format4 = current.strftime("%d-%m-%Y %H:%M:%S") # Y - full digit, y - last 2 digit
17 print("Format 4: ", format4)
18

```

```

Format 1:    2023-09-02 22:21:25
Format 2:    09-02-23 22:21:25
Format 3:    23-02-09 22:21:25
Format 4:    02-09-2023 22:21:25

```

## ✓ Parsing Date and Time

```

1 # @title Parsing Date and Time
2
3 # It is used to perform the date & time calculation when it is in string datatype
4
5 from datetime import datetime
6
7 date_string = "2023-09-01 15:30:00"
8 print(type(date_string))
9
10 parsed_datetime = datetime.strptime(date_string, "%Y-%m-%d %H:%M:%S") # parse function - strptime()
11
12 print(parsed_datetime)
13 print(type(parsed_datetime))

```

```

<class 'str'>
2023-09-01 15:30:00
<class 'datetime.datetime'>

```

## ✓ Timezone Handling

```

1 # @title Timezone Handling
2
3 # for timezone import module 'pytz'
4
5 import pytz
6 from datetime import datetime
7
8 # Coordinated Universal Time or Greenwich Mean Time
9 utc_timezone = pytz.UTC
10 current_datetime = datetime.now(tz=utc_timezone)
11
12 print(current_datetime)
13
14 # USA Eastern time
15 utc_timezone = pytz.timezone('US/Eastern')
16 current = datetime.now(tz=utc_timezone)
17 print(current)
18
19 # India time
20 utc_timezone = pytz.timezone('Asia/Kolkata')
21 current_india = datetime.now(tz=utc_timezone)
22 print(current_india)

```

```

2023-09-02 22:37:14.436878+00:00
2023-09-02 18:37:14.437487-04:00
2023-09-03 04:07:14.440038+05:30

```

## ✓ Arithmetic - Plus

```

1 # @title Arithmetic - Plus
2
3 from datetime import datetime, timedelta
4
5 current_datetime = datetime.now()
6
7 # calculate time for 7 days
8 one_week_later = current_datetime + timedelta(days=7)
9 print("Time and Date for one_week_later: ", one_week_later)
10
11 # calculate time for next 42 hrs
12 next_42_hrs = current_datetime + timedelta(hours = 2)
13 print("Time and Date for next_42_hrs: ", next_42_hrs)

Time and Date for one_week_later: 2023-09-09 22:43:41.245845
Time and Date for next_42_hrs: 2023-09-03 00:43:41.245845

```

## ✖ Minus

```

1 # @title Minus
2
3 from datetime import datetime
4
5 date1 = datetime(2023, 9, 1)
6 date2 = datetime(2023, 8, 15)
7
8 date_difference = date1 - date2
9 print(date_difference.days) # Number of days between the two dates
10

17

```

**Note:** We can't directly perform multiplication and division in the datetime datatype

## ✖ Comparison

```

1 # @title Comparison
2
3
4 from datetime import datetime
5
6 date1 = datetime(2023, 9, 1)
7 date2 = datetime(2023, 8, 15)
8
9 if date1 > date2:
10     print("date1 is later than date2")
11

date1 is later than date2

```

## ✖ Calendar

This module is used to create calendar, manipulate dates, and perform various operations related to date and time.

---

Let see some of the basic function as follow

### ✖ Calendar

```

1 # @title Calendar
2
3 import calendar # import calendar module
4
5 # Create a calendar for the year 2023
6 cal = calendar.calendar(2024)
7
8 # Print the calendar
9 print(cal)
10

```

### ✖ Calendar for Specific Month

```

1 # @title Calendar for Specific Month
2
3 import calendar
4
5 # Create a calendar for November 2023
6 cal = calendar.month(2023, 11)
7
8 # Print the calendar for November
9 print(cal)
10

```

```

      November 2023
Mo Tu We Th Fr Sa Su
                1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26

```

## ▼ Month Names

```

1 # @title Month Names
2
3 # It provide the full month names and abbreviated month names
4
5 import calendar
6
7 # Access the month names and abbreviations
8 print(calendar.month_name[1])    # January
9 print(calendar.month_abbr[1])    # Jan
10

```

```

      January
      Jan

```

## ▼ Weekday for the Secific Date

```

1 # @title Weekday for the Secific Date
2
3 # Note:  0-Monday, 1-Tuesday, 2-Wednesday, 3-Thursday, 4-Friday, 5-Saturday, 6-Sunday
4
5 import calendar
6
7 # Get the weekday of November 30, 2023
8 weekday = calendar.weekday(2023, 11, 30)
9
10 print("Weekday of November 30, 2023:", weekday) # Output: 3 (Thursday)
11

```

```

      Weekday of November 30, 2023: 3

```

## ▼ Check Leap Year

```

1 # @title Check Leap Year
2
3 import calendar
4
5 # Check if 2024 is a leap year
6 is_leap = calendar.isleap(2024) # it returns boolean value
7
8 print("Is 2024 a leap year?", is_leap) # Output: True
9

```

```

      Is 2024 a leap year? True

```