# Modules

- Python as any other programming languages has built-in packages that can be imported and used without being explicitly installed.
- Importing a Module :
- To use the methods of a module, we must import the module into our code. This can be done using the import keyword.
    - import modulename
    - import modulename as md
    - from modulename import methodname, methodname
    - from modulename import *

# Random

- This module implements pseudo-random number generators for various distributions.

```
>>> import random
>>> random.random()
0.645173684807533
```

```
>>> random.randint(1, 100)
95
>>> random.randrange(1, 10)
2
>>> random.choice('computer')
't'
```

```
>>> numbers=[12,23,45,67,65,43]
>>> random.shuffle(numbers)
>>> numbers
[23, 12, 43, 65, 67, 45]
```

- Random.random() returns a float number between 0 and 1 randint(x,y) will return a value >= x and <= y, while randrange(x,y) will return a value >=x and < y (n.b. not less than or equal to y)

# Try, Except

- Python has many built-in exceptions that are raised when your program encounters an error
- (something in the program goes wrong)
- built-in exceptions
    - IOError , ValueError , ImportError , EOFError , KeyboadInterrupt ,ZeroDivisionError…

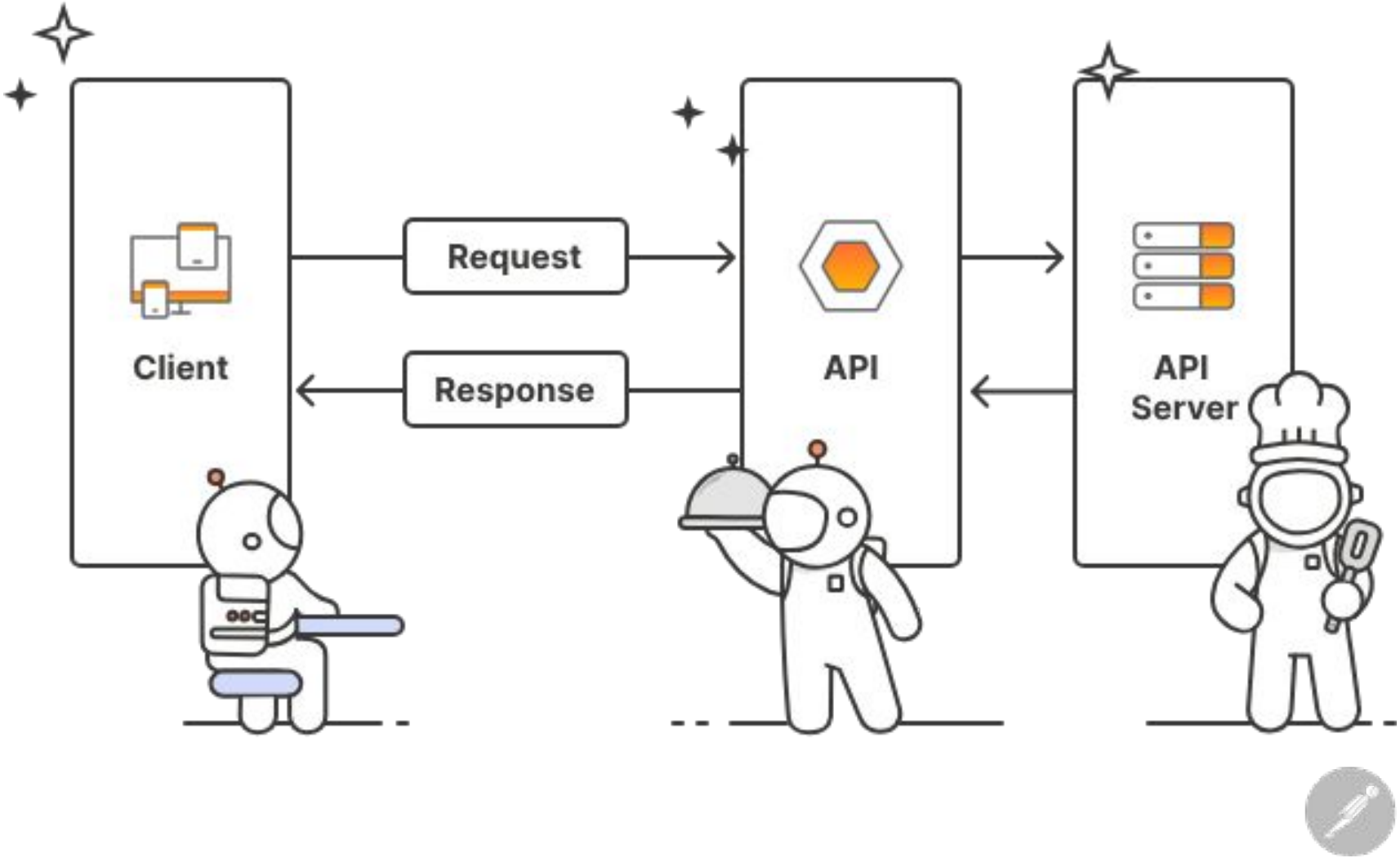
Link to Error Handling

```
try:
  print(x)
except:
  print("An exception occurred")
```

# API

# What is an API?

- API stands for Application Programming Interface.
- It allows different software applications to communicate and share data.
- Works like a translator between two systems, enabling them to work together.


Client
Request
Response
API
API
Server

# API

End User *with Browser*

Request

Response


Server *Back-end System*

Customer

Make the Order

Delivery of order

Waiter

Take the Order

Bringing from Kitchen

Chef

## How APIs Work: Request-Response Cycle

1. **API Client**:
    - The application sending a request.
    - Example: A mobile app asking a server for data.
2. **API Server**:
    - The system that receives the request and processes it.
    - Example: A database or web server that holds the requested information.

# API Request Components

- **Endpoint**: The URL where the API can be accessed (e.g., `/users`).
- **Method**:  The type of request (e.g., `GET`, `POST`, `PUT`, `DELETE`).
- **Parameters**:
    - Information passed to the API,
        - In the URL or the request body.
        - **Or in Body**: Data sent with the request (only for `POST` and `PUT` methods).
- **Headers**: Extra information like authentication or content type.

## API Response Components

- **Status Code**: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status#redirection_messages
  - Indicates the result of the request (e.g., `200 OK`, `404 Not Found`).
- **Headers**:
  - Additional information like content type.
- **Response Body**:
  - The actual data returned by the API or an error message.

# Understanding Common API Status Code

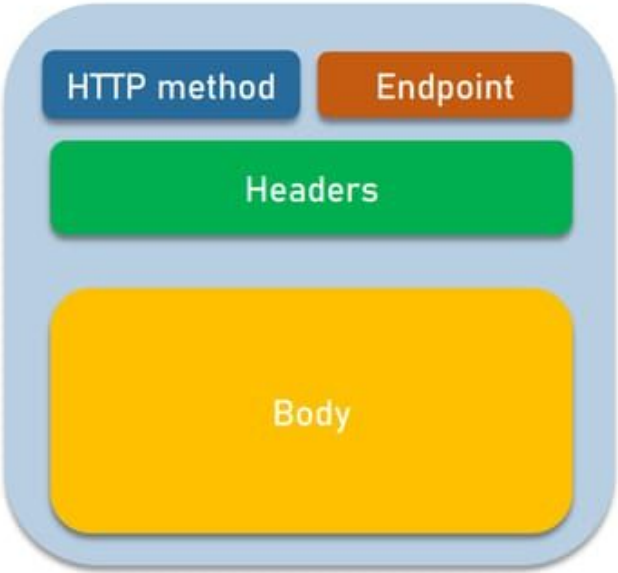
- Every request to a web server returns a status code indicating what happened with the request. Here are some common codes relevant to GET requests:

  - 200: Everything went okay, and the result has been returned (if any).
  - 301: The server is redirecting you to a different endpoint. This can happen when a company switches domain names, or an endpoint name is changed.
  - 400: The server thinks you made a bad request. This happens when you send incorrect data or make other client-side errors.
  - 401: The server thinks you're not authenticated. Many APIs require login credentials, so this happens when you don't send the right credentials to access an API.
  - 403: The resource you're trying to access is forbidden: you don't have the right permissions to see it.
  - 404: The resource you tried to access wasn't found on the server.
  - 503: The server is not ready to handle the request.

# REST API REQUEST

```
POST /api/2.2/sites/9a8b7c6d-5e4f-3a2b-1c0d-9e8f7a6b5c4d/users HTTP/1.1
HOST: my-server
X-Tableau-Auth: 12ab34cd56ef78ab90cd12ef34ab56cd
Content-Type: application/json

{
  "user": {
    "name": "NewUser1",
    "siteRole":  "Publisher"
  }
}
```




altexsoft

## HTTP METHOD

**POST**

## REQUEST URL

Path Parameter

http://<yourServerHost>/fmerest/v3/projects/FME_PROJECT_TEST/export/download?accept=contents

Query String Parameter

## REQUEST HEADER

**Content-Type: application/x-www-form-urlencoded**

**Accept: application/json**

## REQUEST BODY

**excludeSensitiveInfo=false&exportPackageName=ProjectPackage.fsproject**

# REST API RESPONSE WITH HYPERMEDIA

```
HTTP/1.1 200 OK
Content-Type: text/html

<!DOCTYPE html>
<html>
  <head>
    <title>Home Page</title>
  </head>
  </body>
    <div>Hello World!</div>
    <a href= "http://www.recurse.com"> Check out the Recurse Center! </a>
    <img src="awesome-pic.jpg">
  </body>
</html>
```

tells the client to make a GET request to http://www.example.com/awesome-pic.jpg to display the user's image

tells the client to make a GET request to http://www.recurse.com if the user clicks on the link

altexsoft

## Example of API in Action

- **User** searches for a product in an e-commerce app.
- The **API Client** (app) sends a `GET` request to the server.
- The **API Server** returns product details as a **Response**.
- The **Client** displays the results to the user.

# API architectural styles

- **REST API**: A simple and widely-used design pattern for communicating over HTTP between applications using standard methods like GET and POST.
- **FastAPI**: A modern, very fast Python framework for building APIs, offering automatic documentation and high performance.
- **GraphQL**: A flexible query language that allows you to request exactly the data you need from a server.
- **gRPC**: A high-performance framework by Google that uses HTTP/2 to efficiently exchange data between services.

# API architectural styles

- **SOAP**: An older and reliable protocol that uses XML to transfer data, ideal for sensitive systems like banking.
- **JSON-RPC**: A lightweight protocol based on JSON for performing remote procedure calls between applications.
- **OData**: A protocol that uses HTTP to query and manipulate data in a way similar to SQL.
- **OpenAPI/Swagger**: A standard for documenting and designing REST APIs with an interactive interface for easier understanding and use.

# Making API Requests

- If you use pip to manage your Python packages, you can install the requests library using the following command:

```
pip install requests
```

- Once you've installed the library, you'll need to import it. Let's start with that important step:

```
import requests
```

# Introduction to REST APIs

**REST (Representational State Transfer):**

- Architectural style for designing networked applications
- Uses a stateless communication protocol, typically HTTP

# Key Principles of REST

**Statelessness:**
Each request from a client to a server must contain all the information needed to understand and process the request

**Client-Server Architecture:**
Separation of client and server concerns

**Cacheability:**
Responses must define themselves as cacheable or not to prevent clients from reusing stale data

**Uniform Interface:**
Simplifies and decouples the architecture, allowing each part to evolve independently

**Layered System:**
A client cannot ordinarily tell whether it is connected directly to the end server or to an intermediary along the way

# Files I/O

# Files I/O

The simplest way to produce output is using the *print* statement where you can pass zero or more expressions separated by commas. This function converts the expressions you pass into a string and writes the result to standard output as follows:

```
Print("Python is really a great language,", "isn't it?“)
```

This produces the following result on your standard screen

```
Python is really a great language, isn't it?
```

## Reading Keyboard Input

Python provides two built-in functions to read a line of text from standard input, which by default comes from the keyboard. These functions are

- raw_input
- input

## The *raw_input* Function

The *raw_input([prompt])* function reads one line from standard input and returns it as a string (removing the trailing newline).

```
str = raw_input("Enter your input: ");
print "Received input is : ", str
```

This prompts you to enter any string and it would display same string on the screen. When I typed "Hello Python!", its output is like this

```
Enter your input: Hello Python
Received input is :  Hello Python
```



## The input Function

The input([prompt]) function is equivalent to raw_input, except that it assumes the input is a valid Python expression and returns the evaluated result to you.

```
str = input("Enter your input: ");
print("Received input is : ", str)
```

This would produce the following result against the entered input

```
Enter your input: [x*5 for x in range(2,10,2)]
Received input is :  [10, 20, 30, 40]
```



# Opening and Closing Files

Until now, you have been reading and writing to the standard input and output. Now, we will see how to use actual data files.

Python provides basic functions and methods necessary to manipulate files by default. You can do most of the file manipulation using a **file** object.



# The open Function

Before you can read or write a file, you have to open it using Python's builtin open() function. This function creates a **file** object, which would be utilized to call other support methods associated with it.

Syntax

```
file object = open(file_name [, access_mode][, buffering])
```



# The open Function

Here are parameter details:

- **file_name:** The file_name argument is a string value that contains the name of the file that you want to access.
- **access_mode:** The access_mode determines the mode in which the file has to be opened, i.e., read, write, append, etc. A complete list of possible values is given below in the table. This is optional parameter and the default file access mode is read (r).



## The open Function

- **buffering:** If the buffering value is set to 0, no buffering takes place. If the buffering value is 1, line buffering is performed while accessing a file. If you specify the buffering value as an integer greater than 1, then buffering action is performed with the indicated buffer size. If negative, the buffer size is the system default(default behavior).



## Here is a list of the different modes of opening a file:

| Modes | Description |
|---|---|
| r | Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode. |
| rb | Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode. |
| r+ | Opens a file for both reading and writing. The file pointer placed at the beginning of the file. |
| rb+ | Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file. |
| w | Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing. |
| wb | Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing. |



| | |
|---|---|
| w+ | Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing. |
| wb+ | Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing. |
| a | Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing. |
| ab | Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing. |
| a+ | Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing. |
| ab+ | Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing. |



# The file Object Attributes

Once a file is opened and you have one file object, you can get various information related to that file.

Here is a list of all attributes related to file object:

| Attribute | Description |
|---|---|
| file.closed | Returns true if file is closed, false otherwise. |
| file.mode | Returns access mode with which file was opened. |
| file.name | Returns name of the file. |
| file.softspace | Returns false if space explicitly required with print, true otherwise. |



## Example

```
# Open a file
fo = open("foo.txt", "wb")
print "Name of the file: ", fo.name
print "Closed or not : ", fo.closed
print "Opening mode : ", fo.mode
print "Softspace flag : ", fo.softspace
```

```
Name of the file:  foo.txt
Closed or not :  False
Opening mode :  wb
Softspace flag :  0
```



# The close() Method

The close() method of a file object flushes any unwritten information and closes the file object, after which no more writing can be done.

Python automatically closes a file when the reference object of a file is reassigned to another file. It is a good practice to use the close() method to close a file.

Syntax

```
fileObject.close();
```



## Example

```
# Open a file
fo = open("foo.txt", "wb")
print "Name of the file: ",
fo.name

# Close opend file
fo.close()
```

```
Name of the file:  foo.txt
```



# The write() Method

The write() method writes any string to an open file. It is important to note that Python strings can have binary data and not just text.

The write() method does not add a newline character ('\n') to the end of the string –

Syntax

```
fileObject.write(string);
```

Here, passed parameter is the content to be written into the opened file.



## Example

```
# Open a file
fo = open("foo.txt", "wb")
fo.write( "Python is a great
language.\nYeah its great!!\n");

# Close opend file
fo.close()
```

```
Python is a great language.

Yeah its great!!
```



# The read() Method

The read() method reads a string from an open file. It is important to note that Python strings can have binary data. apart from text data.

Syntax

```
fileObject.read([count]);
```

Here, passed parameter is the number of bytes to be read from the opened file. This method starts reading from the beginning of the file and if count is missing, then it tries to read as much as possible, maybe until the end of file.



## Example

```
# Open a file
fo = open("foo.txt", "r+")
str = fo.read(10);
print "Read String is : ", str
# Close opened file
fo.close()
```

```
Read String is :  Python is
```



# Directories in Python

All files are contained within various directories, and Python has no problem handling these too.

The **os** module has several methods that help you create, remove, and change directories.



# The mkdir() Method

You can use the mkdir() method of the **os** module to create directories in the current directory. You need to supply an argument to this method which contains the name of the directory to be created.

Syntax

```
os.mkdir("newdir")
```

```
#!/usr/bin/python
import os

# Create a directory "test"
os.mkdir("test")
```



# The chdir() Method

You can use the chdir() method to change the current directory. The chdir() method takes an argument, which is the name of the directory that you want to make the current directory.

Syntax

```
os.chdir("newdir")
```

```
import os

# Changing a directory to "/home/newdir"
os.chdir("/home/newdir")
```



# The getcwd() Method

The getcwd() method displays the current working directory.

Syntax

```
os.getcwd()
```

```
import os

# This would give location of the current directory
os.getcwd()
```



The rmdir() Method

The rmdir() method deletes the directory, which is passed as an argument in the method.

Before removing a directory, all the contents in it should be removed.

Syntax:

```
os.rmdir('dirname')
```

```
import os

# This would  remove "/tmp/test"  directory.
os.rmdir( "/tmp/test"  )
```