MDMC
Master in Data Management and Curation

AREA SCIENCE PARK

SISSA

Advanced Python Programming
Basic Software Development

Marco Prenassi
Laboratory of Data Engineering, Area Science Park
date

TABLE OF CONTENTS:

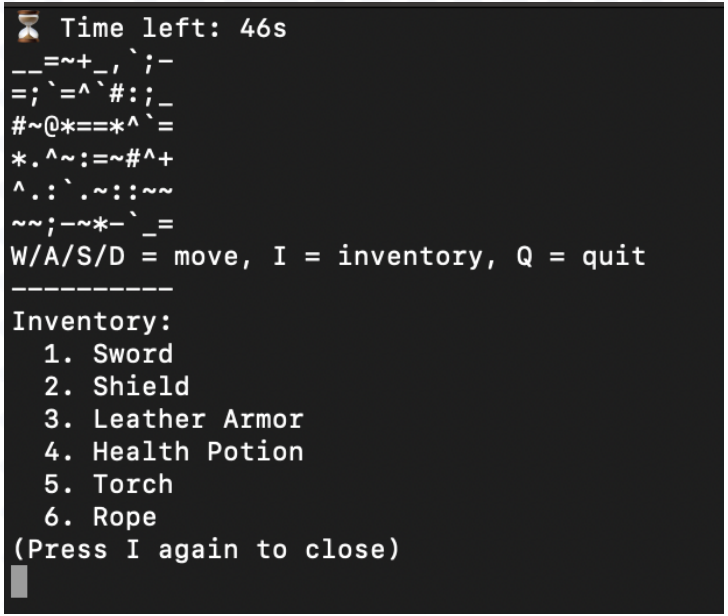
- Why?
- Basic software development concepts
- Basic approaches to development

SECONDARY GOALS:

- Work as a team

WHY?

- Let’s make a simple game

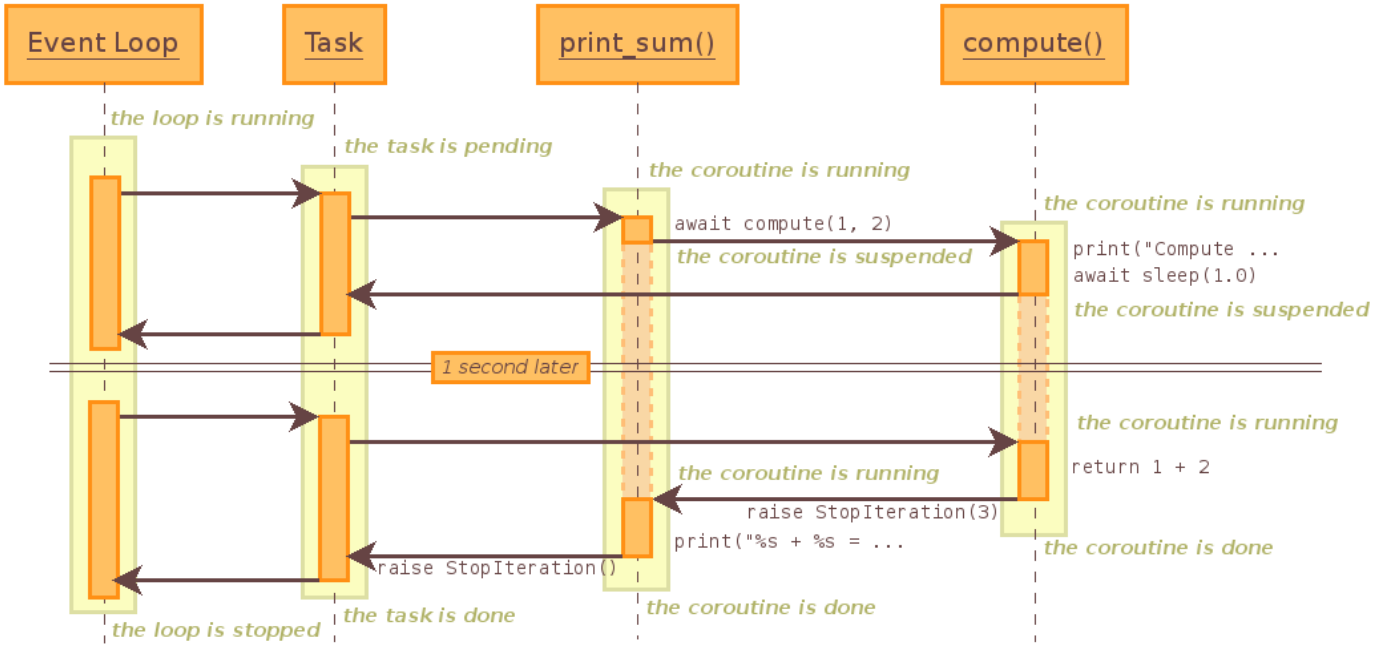


GITHUB:

git clone https://github.com/Master-Data-Management-and-Curation/Advanced-Python-2025.git

WHY?

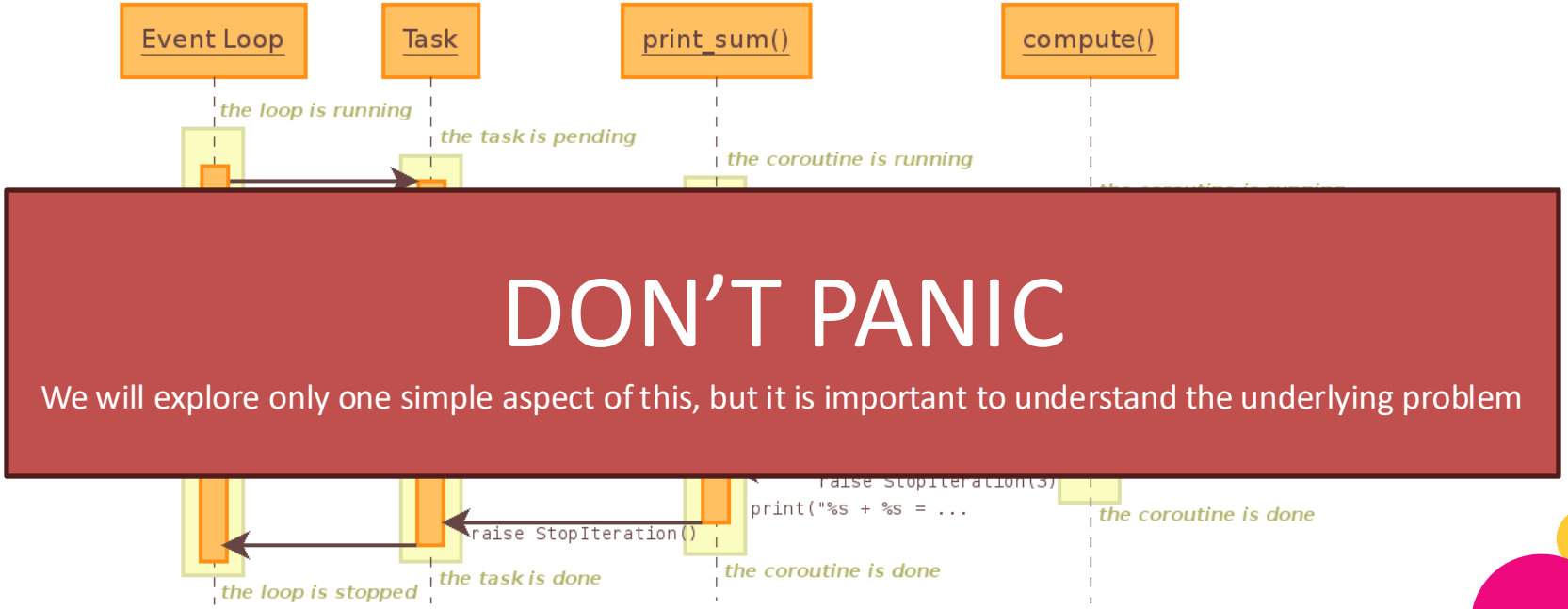
- **Managing Events (1/2)**



https://docs.python.org/id/3.5/library/asyncio-task.html

WHY?

- **Managing Events**



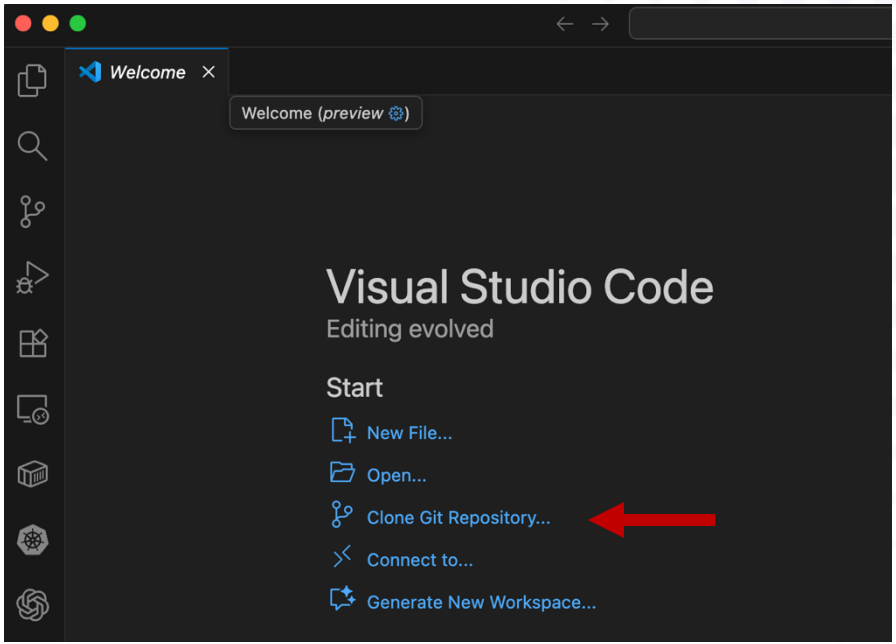
DON'T PANIC

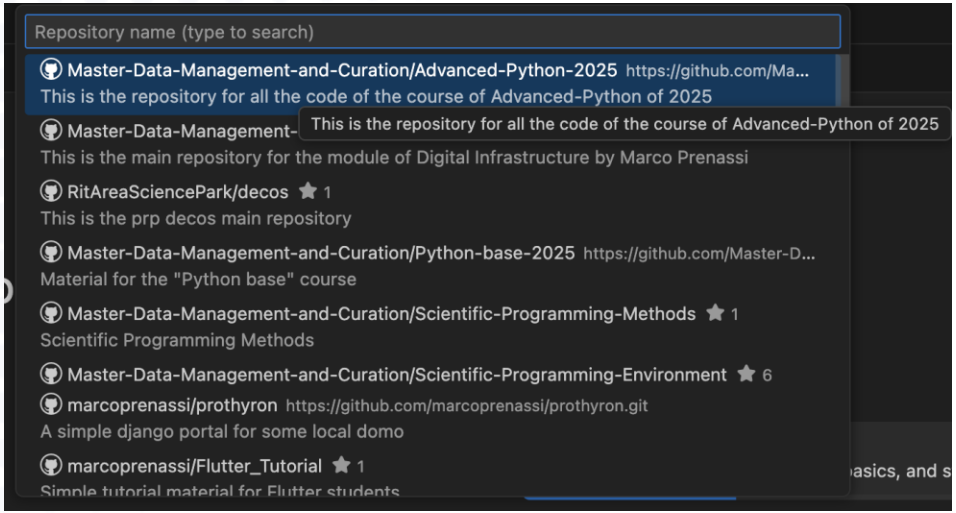
We will explore only one simple aspect of this, but it is important to understand the underlying problem

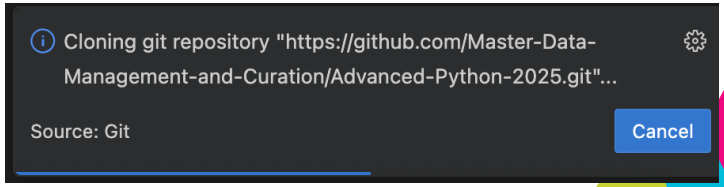
https://docs.python.org/id/3.5/library/asyncio-task.html

IDE: INTEGRATED DEVELOPMENT ENVIRONMENT

e.g., VISUAL STUDIO CODE

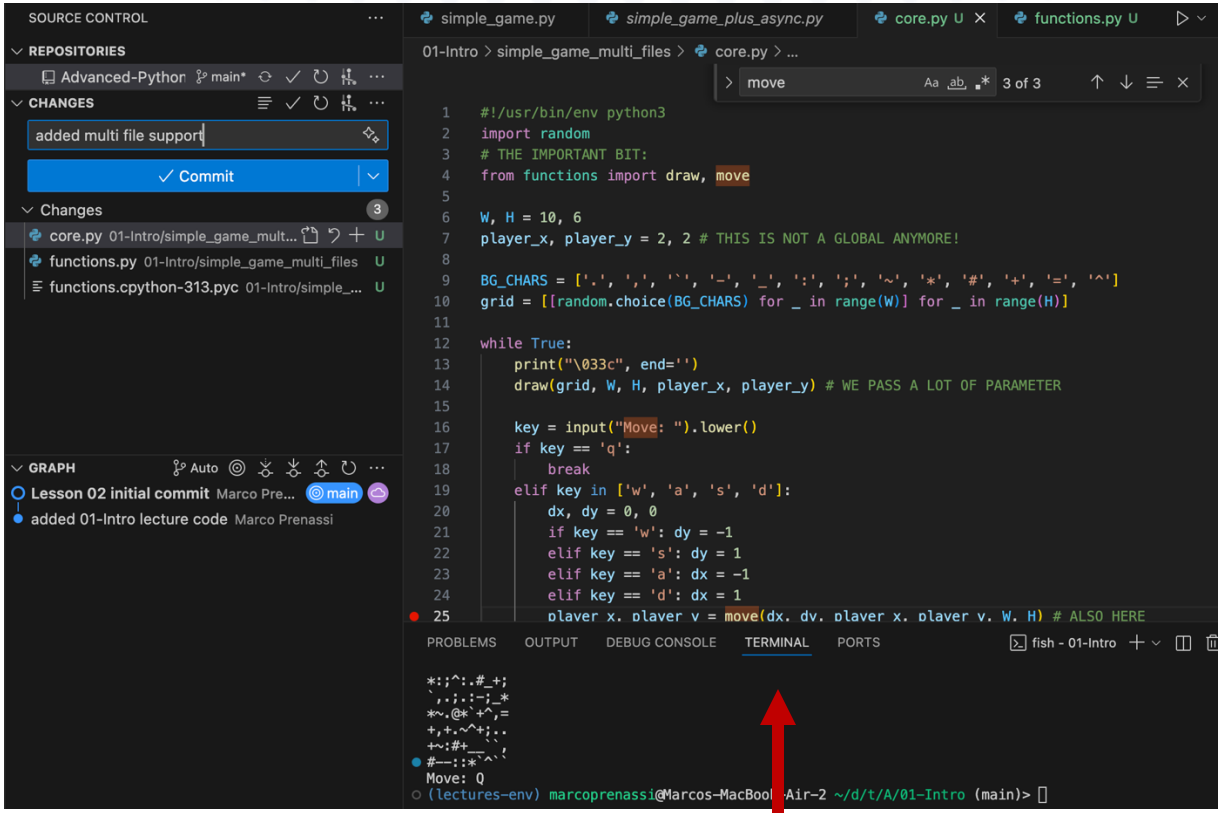






https://code.visualstudio.com/

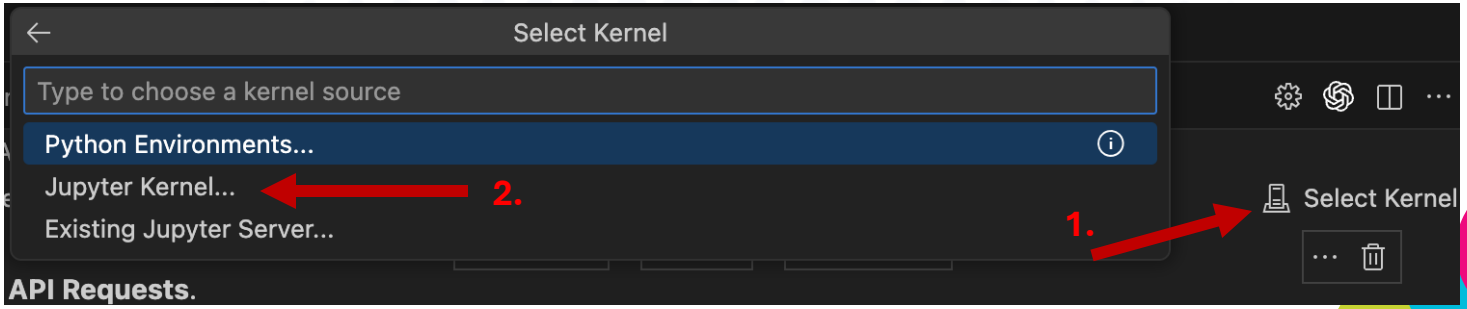
IDE: WHY



https://code.visualstudio.com/

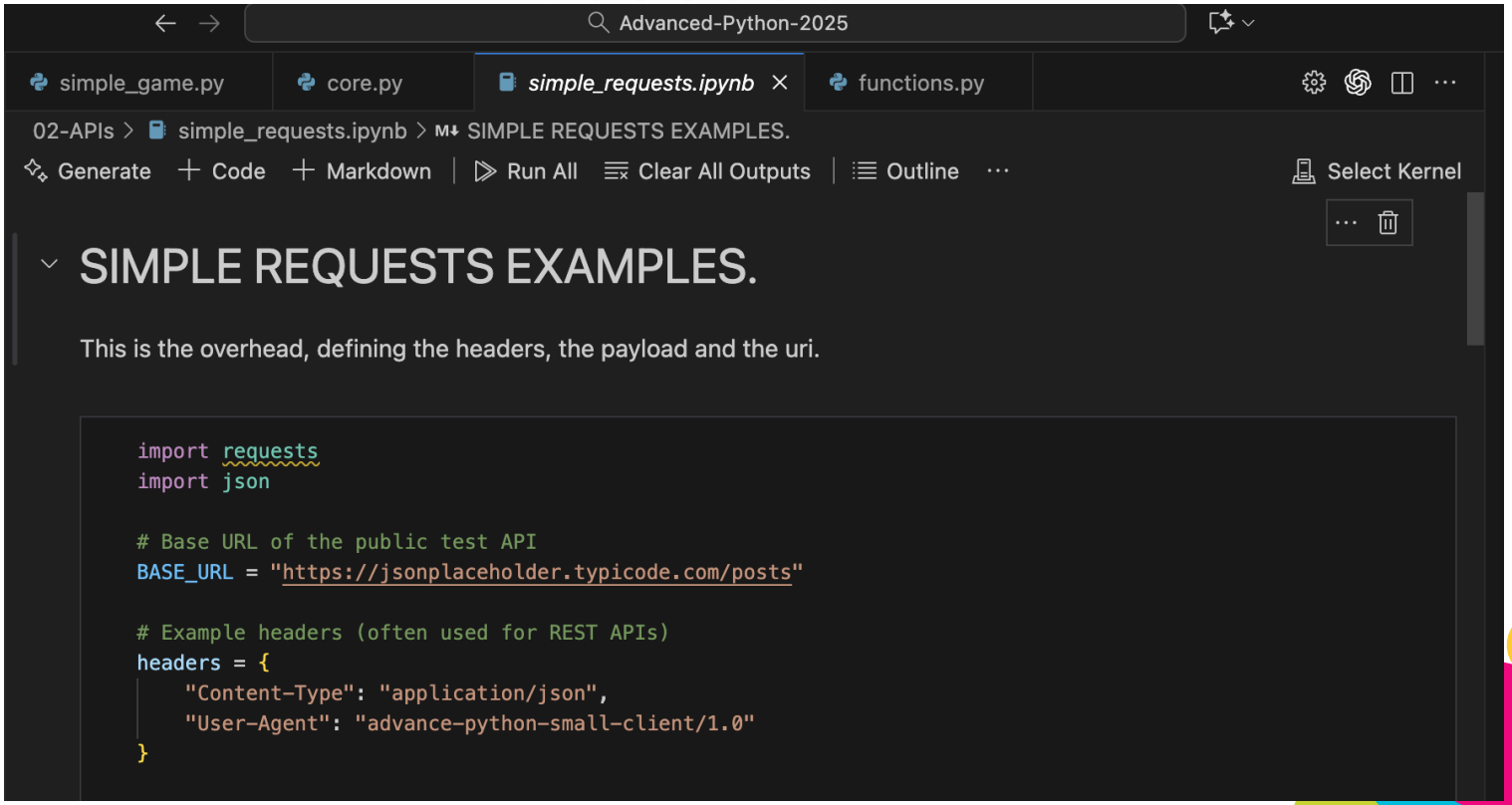
IDE: PYTHON and JUPYTER NOTEBOOKS

1. Create or activate the environment you want the kernel to use (in the folder of the repository)
   ```
   python3 -m venv lecture-env
   ```
   ```
   source ./lecture-env/bin/activate
   ```
2. Install the kernel machinery in that env:
   ```
   pip install ipykernel
   ```
3. Register the kernel with Jupyter, giving it a recognizable name:
   ```
   python -m lecture-env install --user --name advanced-python-kernel --display-name "Advanced Python Kernel"
   ```
4. Restart VSCODE, open a jupyter notebook and then select:



https://code.visualstudio.com/

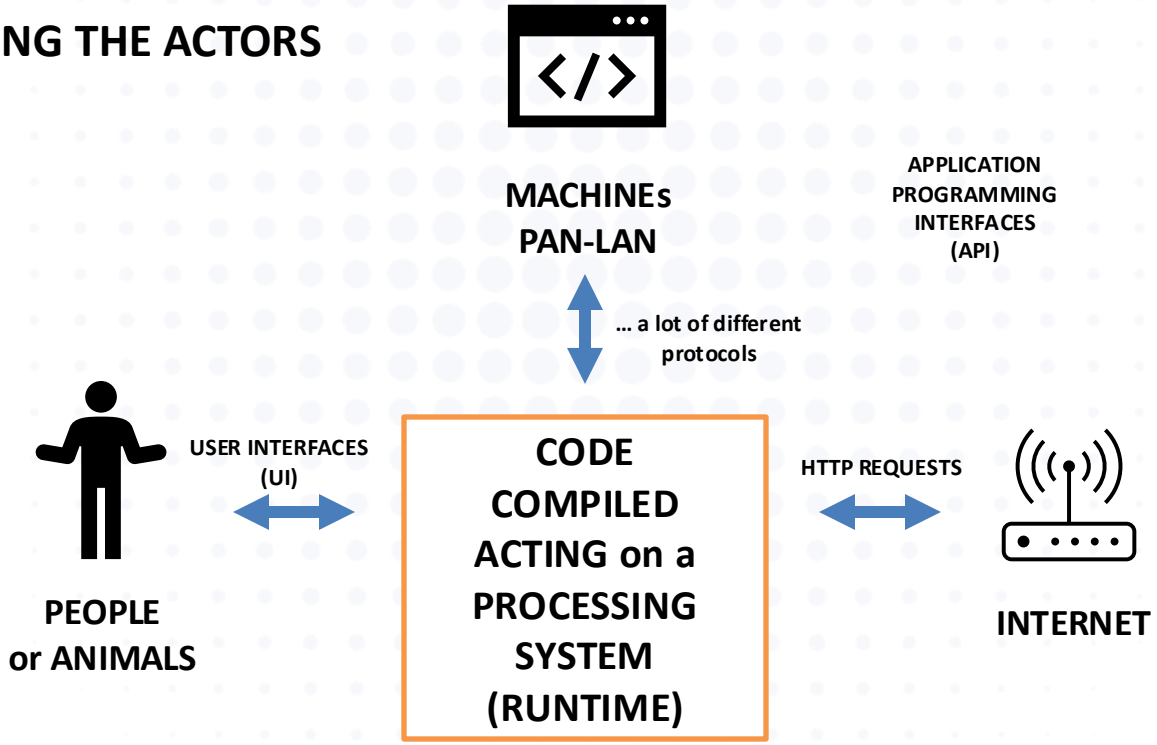
IDE: PYTHON and JUPYTER NOTEBOOKS



https://code.visualstudio.com/

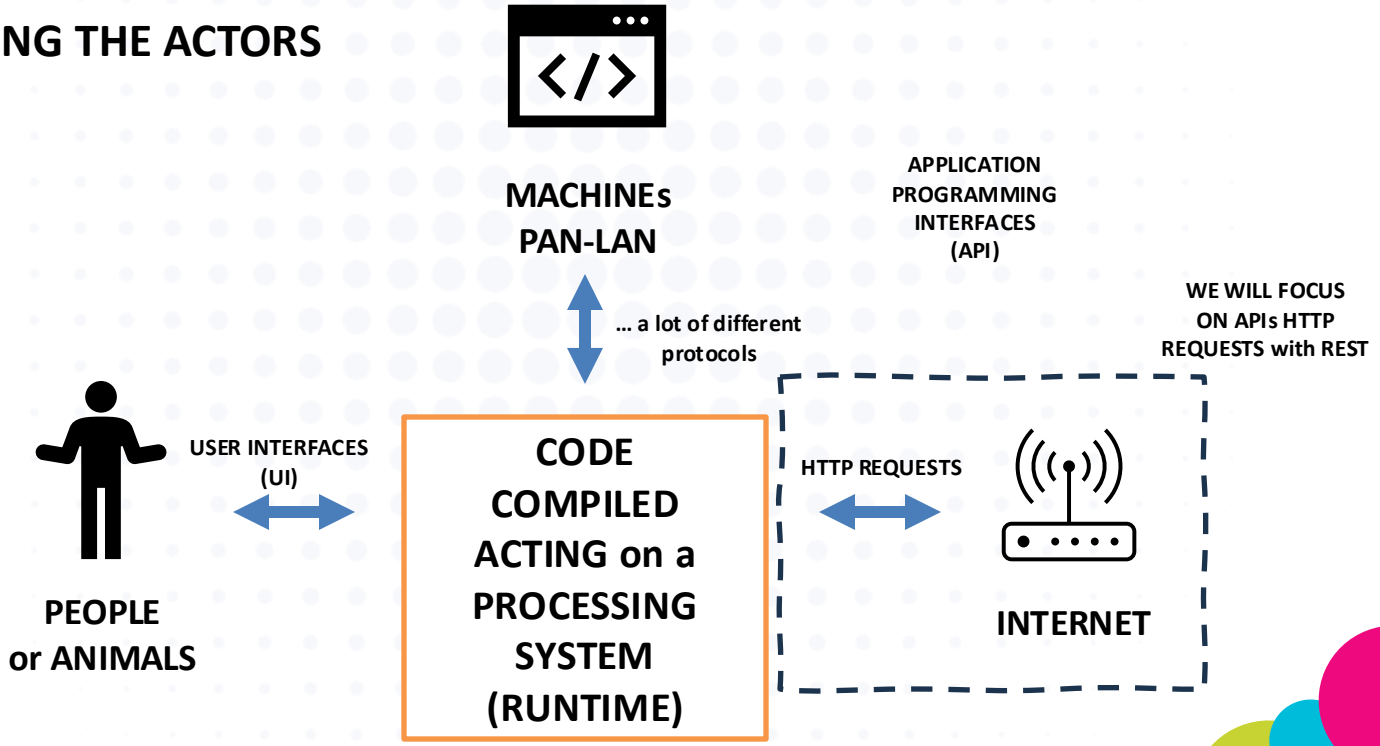
MANAGING EVENTS

INTRODUCING THE ACTORS



MANAGING EVENTS

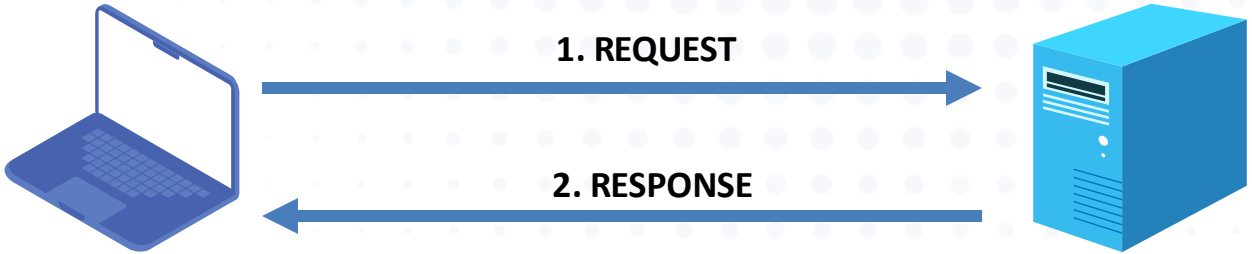
INTRODUCING THE ACTORS



HTTP PROTOCOL BASIC CONCEPTS

HTTP (HyperText Transfer Protocol) is **standard application-level protocol** used for exchanging **files on the World Wide Web**.
HTTP runs on top of the TCP/IP protocol and (later) on the QUIC protocol.
Web browsers are HTTP clients that send file requests to **Web servers**, which in turn handle the requests via an **HTTP service**.

STATELESS!!
EVERY REQUEST IS NOT DEFINED BY THE PREVIOUS ONE



https://www.britannica.com/technology/HTTP

HTTP REQUESTS

An HTTP request has **four main parts**:

1.Method (Type): what action to perform

2.URL (Endpoint): what resource to act on

3.Headers: extra info about the request (e.g., Authorization)

4.Body (Payload): data sent to the server (if needed, remember, these are REQUESTS)

e.g.,
1. POST /api/users HTTP/1.1
2. Host: api.example.com
3. Content-Type: application/json
4. { "name": ”John", "role": ”researcher" }

Request method	RFC	Request has payload body	Response has payload body
GET	RFC 9110	Optional	Yes
HEAD	RFC 9110	Optional	No
POST	RFC 9110	Yes	Yes
PUT	RFC 9110	Yes	Yes
DELETE	RFC 9110	Optional	Yes
CONNECT	RFC 9110	Optional	Yes
OPTIONS	RFC 9110	Optional	Yes
TRACE	RFC 9110	No	Yes
PATCH	RFC 5789	Yes	Yes

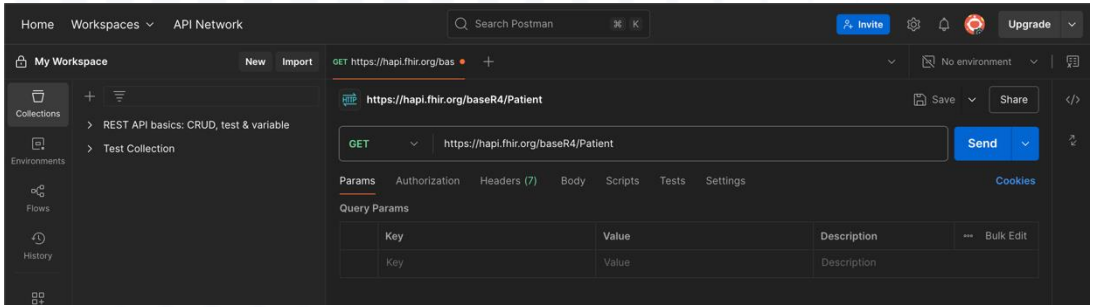
https://en.wikipedia.org/wiki/HTTP

HTTP REQUESTS – LET'S TRY

HOW:

1. **CURL (UNIX or WINDOWS ALIAS)**
2. **REQUESTS MODULE IN PYTHON**
3. **POSTMAN**
4. **… MANY MORE**

POSTMAN (https://web.postman.co/home):



CURL:

Unix (macOs + Linux):

curl --method GET -H "Accept: application/fhir+json" https://hapi.fhir.org/baseR4/Patient/example

Windows powershell:

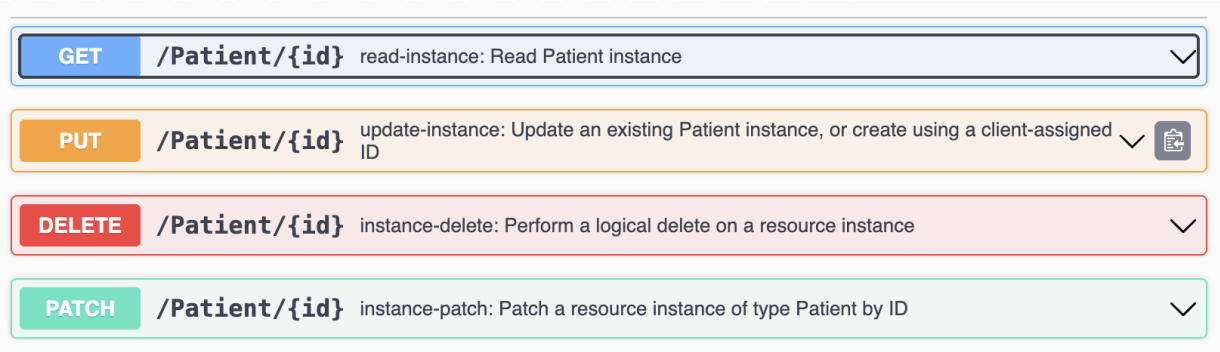
Invoke-RestMethod -Uri "https://hapi.fhir.org/baseR4/Patient/example" -Method GET
-Headers @{ "Accept" = "application/fhir+json" } (In a single line!!!)

HTTP REQUESTS - SWAGGER

Swagger:
https://hapi.fhir.org/baseR4/swagger-ui/

Swagger allows you to describe the structure of your APIs so that machines can read them. The ability of APIs to describe their own structure is the root of all awesomeness in Swagger.

from https://swagger.io/docs/specification/v2_0/what-is-swagger/



HTTP REQUESTS – RESPONSE CODES

1xx	2xx STATUS CODES	3xx	4xx CLIENT ERRORS	5xx SERVER ERRORS
	200 Ok 201 Accepted …		400 Bad Request 401 Unauthorized … 403 Forbidden 404 Not Found … 418 I'm a teapot …	500 Internal Server Error … 502 Bad Gateway 503 Service Unavailable …

HTTP REQUESTS – RESPONSE CODES

LET'S TRY SOMETHING SIMPLE WITH: 02-APIs/simple_requests.ipynb

snippets of code:

Create custom headers:

```
headers = {
     "Authorization": "Bearer your_api_token", "Content-Type": "application/json",
     "User-Agent": "Python-Requests/Example"
     }
```

Data to send with the POST request:

```
post_data = { "key1": "value1", "key2": "value2" }
```

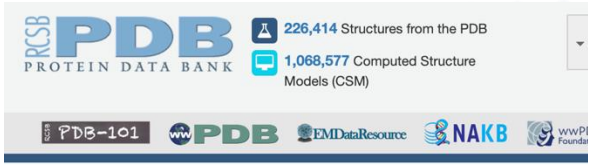
Perform the POST request :

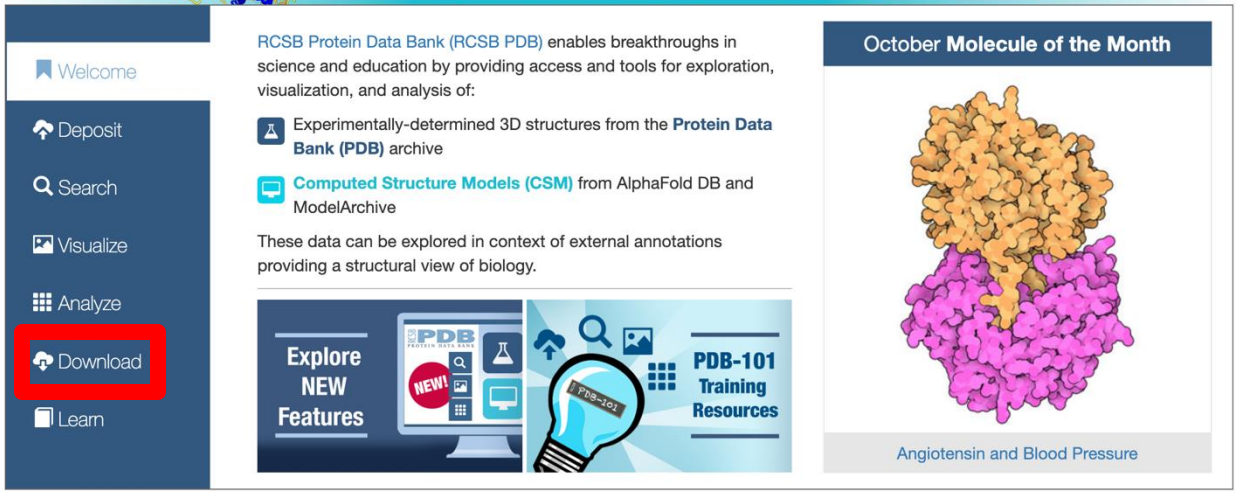
```
try:
     post_response = requests.post(post_url, json=post_data, headers=headers)
     print(f"Status Code: {post_response.status_code}")
     print(f"Response Body: {post_response.json()}")
except requests.RequestException as e:
     print(f"Error during POST request: {e}")
```

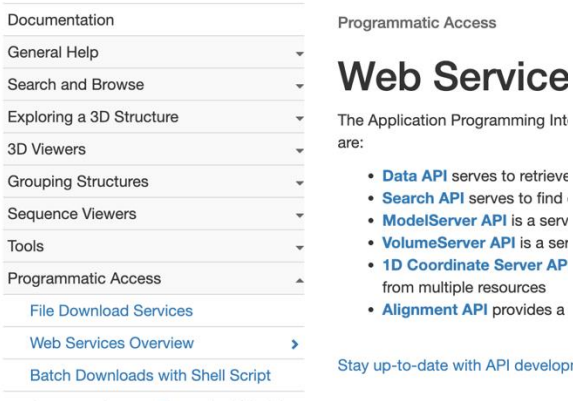
HTTP REQUESTS – LET'S TRY ON SOMETHING REAL

LET'S START WITH RCSB

a







HTTP REQUESTS – LET'S TRY ON SOMETHING REAL

LET'S START WITH RCSB

and jupyter notebooks on

`02-APIs/RCSB_Mapping.ipynb`

RESTful

- **Uniform interface**: One piece of data belong to a single URI
- **Client-server decoupling**: You only need to know the URI of the server to interact
- **Statelessness**: all the information to process is included in the message, no previous operations required
- **Cacheability**: everything that is cacheable must be cached, to improve performance (client AND server side)
- **Layered system architecture**: the architecture is transparent to the layers inside
- **Code on demand***: could give runnable code as a response (this is risky and must be done only on-demand).

SOAP vs REST

Simple Object Access Protocol (SOAP)! → POST ← GET, xml is an envelope

```
HTTP POST /ReturnValue HTTP/1.0 Host: www.abc.net Content-Type: text/xml; charset = utf-8 Content-Length: nnn
<?xml version = "1.0"?> <SOAP-ENV:Envelope xmlns:SOAP-ENV = "http://www.w3.org/2001/12/soap-envelope"
SOAP-ENV:encodingStyle = "http://www.w3.org/2001/12/soap-encoding">
<SOAP-ENV:Body xmlns:m = "http://www.abc.net/values"> <m:Value> <m:ValueName>Temperature</m:ValueName>
</m:GetValue> </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

HTTP/1.0 200 OK Content-Type: text/xml; charset = utf-8 Content-Length: nnn
<?xml version = "1.0"?> <SOAP-ENV:Envelope xmlns:SOAP-ENV = "http://www.w3.org/2001/12/soap-envelope"
SOAP-ENV:encodingStyle = "http://www.w3.org/2001/12/soap-encoding">
<SOAP-ENV:Body xmlns:m = "http://www.abc.net/values"> <m: ValueResponse > <m: Value >25.0</m: Value>
</m: ValueResponse > </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

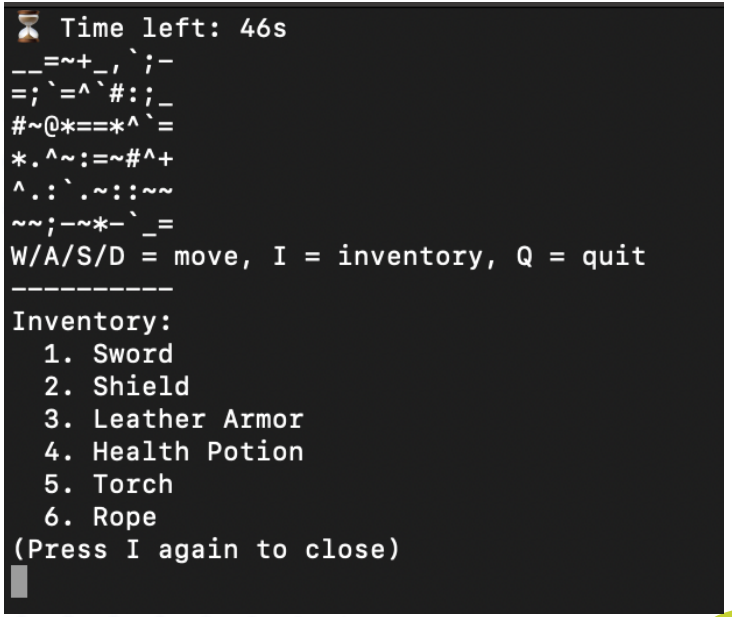
How to "move fast": HTTP GET http://www.abc.net?value=temperature -> Response...

With **RESTful**: HTTP GET http://www.abc.net/values/temperature -> 25.0 (in JSON: {"temperature": 25.0})
also: let's use all the commands available in HTTP

WHY?

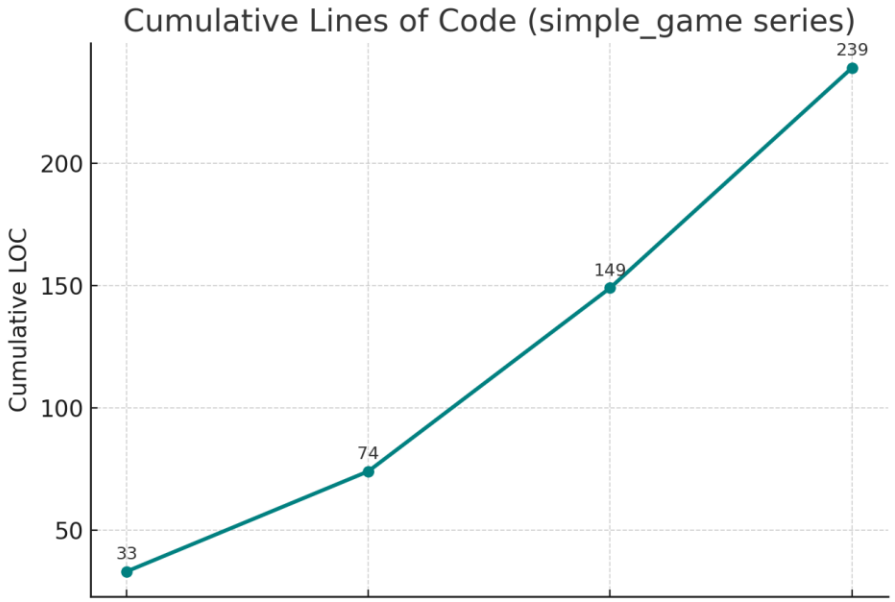
- Let’s make a simple game

GITHUB:
git clone ...



WHY?

- Lines Of Codes (LOC) growth by feature:



WHY?

- Managing complexity (2/2)

1. **Object-Oriented Programming (OOP)**
   Organizes code around objects that combine data and behaviour; promotes encapsulation and reuse.
2. **Functional Programming (FP)**
   Builds programs from pure, stateless functions and immutable data.
3. **Data-Oriented Design (DOD)**
   Focuses on data layout and usage patterns for clarity and performance.
4. **Entity-Component-System (ECS)**
   Composes entities from independent data components and processing systems.
5. **Procedural / Modular Programming**
   Structures programs as reusable functions grouped into modules.
6. **Declarative Programming**
   Describes *what* to do, leaving the *how* to the underlying system.
7. **Actor Model**
   Uses independent actors that communicate via messages to handle concurrency safely.
8. **Reactive Programming**
   Represents logic as data streams that automatically propagate changes.
9. **Aspect-Oriented Programming (AOP)**
   Isolates cross-cutting concerns like logging or security into separate aspects.
10. **Agent-Based Systems**
    Models systems as interacting autonomous agents with their own goals.
11. **Dataflow Programming**
    Defines computation as a directed graph of data dependencies and transformations.

THESE ARE NOT ENEMIES
MULTI-PARADIGM
IS THE NORM

WHY?

- **Managing complexity**

1. **Object-Oriented Programming (OOP)**
   Organizes code around objects that combine data and behaviour; promotes encapsulation and reuse.
2. **Functional Programming (FP)**
   Builds programs from pure, stateless functions and immutable data.
3. **Data-Oriented Design (DOD)**
   Focuses on data layout and usage patterns for clarity and performance.
4. **Entity-Component-System (ECS)**
   Composes entities from independent data components and processing systems.
5. **Procedural / Modular Programming**
   Structures programs as reusable functions grouped into modules.
6. **Declarative Programming**
   Describes *what* to do, leaving the *how* to the underlying system.
7. **Actor Model**
   Uses independent actors that communicate via messages to handle concurrency safely.
8. **Reactive Programming**
   Represents logic as data streams that automatically propagate changes.
9. **Aspect-Oriented Programming (AOP)**
   Isolates cross-cutting concerns like logging or security into separate aspects.
10. **Agent-Based Systems**
    Models systems as interacting autonomous agents with their own goals.
11. **Dataflow Programming**
    Defines computation as a directed graph of data dependencies and transformations.

THESE ARE NOT ENEMIES
MULTI-PARADIGM
IS THE NORM

“OPERATOR OVERLOAD” in PYTHON

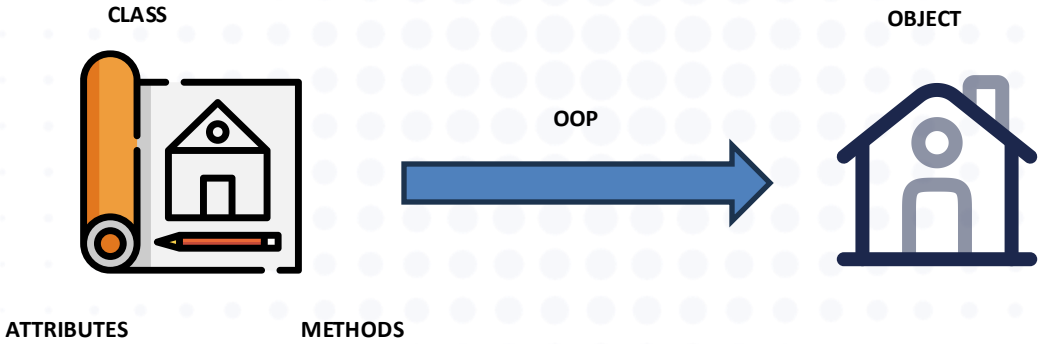
Dunder Methods
special “magic” functions like __init__, __str__, __add__ — they define how objects behave with Python’s built-in syntax.
called automatically at runtime, never by the compiler.

Operator Overloading
in python “overloading” means redefining what +, >, ==, etc. do for your objects through **dunder methods**.
Remember: Python is “simple”, nothing happens at compile time, every operator is just a function call under the hood.

Operator	Method Name	Usage Example
+	__add__(self, other)	a + b
-	__sub__(self, other)	a - b
*	__mul__(self, other)	a * b
/	__truediv__(self, other)	a / b
==	__eq__(self, other)	a == b
len()	__len__(self)	len(obj)
[]	__getitem__(self, key)	obj[key]
in	__contains__(self, item)	item in obj
str()	__str__(self)	print(obj)
repr()	__repr__(self)	repr(obj)
call()	__call__(self, *args)	obj()
del	__del__(self)	del obj

OBJECT ORIENTED PROGRAMMING
WHY IS IT USED SO MUCH?

OBJECT-ORIENTED PARADIGM



OBJECT LIFECYCLE in PYTHON

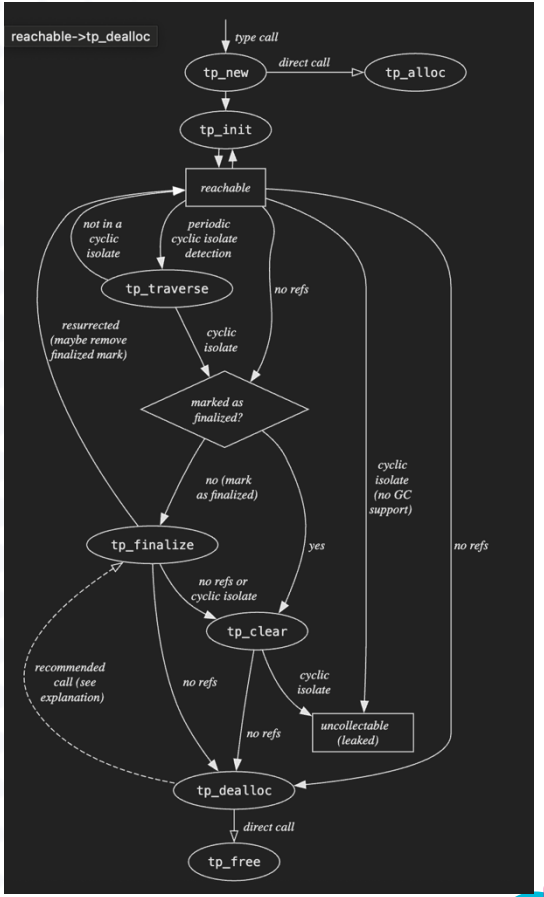
`object.__new__(cls[, ...])`

Called to create a new instance of class *cls*. __new__() is a static method (special-cased so you need not declare it as such) that takes the class of which an instance was requested as its first argument. The remaining arguments are those passed to the object constructor expression (the call to the class). The return value of __new__() should be the new object instance (usually an instance of *cls*). […]

`object.__init__(self[, ...])`
Called after the instance has been created (by __new__()), but before it is returned to the caller. The arguments are those passed to the class constructor expression. If a base class has an __init__() method, the derived class's __init__() method, if any, must explicitly call it to ensure proper initialization of the base class part of the instance; for example: super().__init__([args…]). […]

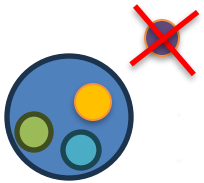
`object.__del__(self)`
Called when the instance is about to be destroyed. This is also called a finalizer or (improperly) a destructor. If a base class has a __del__() method, the derived class's __del__() method, if any, must explicitly call it to ensure proper deletion of the base class part of the instance. […]

```
class Overload_Me:
      def __new__(cls): print("Creating instance");
      return super().__new__(cls)

      def __init__(self): print("Initializing instance")

      def __del__(self): print("Destroying instance")

overload_me_object = Overload_Me()
del overload_me_object
```

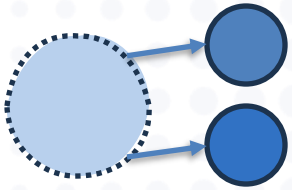


https://docs.python.org/3/c-api/lifecycle.html

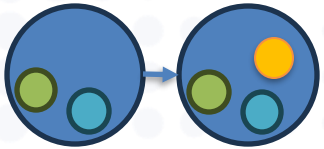
OBJECT AND CLASSES BASIC PROPERTIES



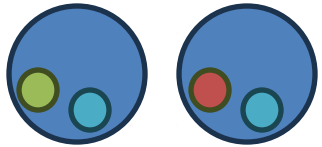
Encapsulation



Abstraction

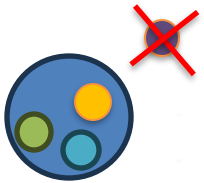


Inheritance



Polymorphism

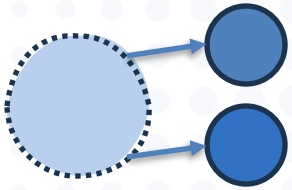
OBJECT AND CLASSES BASIC PROPERTIES



Encapsulation

EVERYTHING IS CONTAINED
INSIDE AN OBJECT

OBJECT AND CLASSES BASIC PROPERTIES



Abstraction

THE FUNCTION COULD BE DESCRIBED REGARDLESS OF THE IMPLEMENTATION

IF I NEED A FUNCTION TO TALK TO PEOPLE OVER LONG DISTANCES

Uberpruiser - Own work

•CC BY-SA 3.0
•File:FTTR RTT56B Telephone 2.jpg
•Created: 16 December 2012
•Uploaded: 17 December 2012

Gustavo Belemmi - Own work

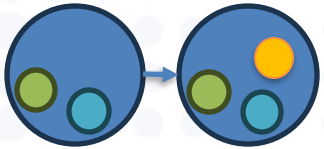
•CC BY-SA 4.0
•File:Antique toy walkie-talkie.jpg
•Created: 13 July 2021
•Uploaded: 14 July 2021

© Raimond Spekking / CC BY-SA 4.0 (via Wikimedia Commons)

•CC BY-SA 4.0
•File:Dell Streak-9014.jpg
•Created: 22 June 2020
•Uploaded: 28 June 2020

OBJECT AND CLASSES BASIC PROPERTIES

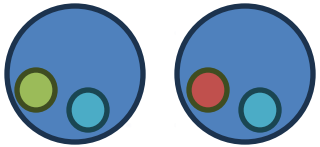
I CAN EXTEND THE FUNCTIONALITIES OF A CLASS



Inheritance

OBJECT AND CLASSES BASIC PROPERTIES

I CAN OVERRIDE THE FUNCTIONALITIES OF A CLASS



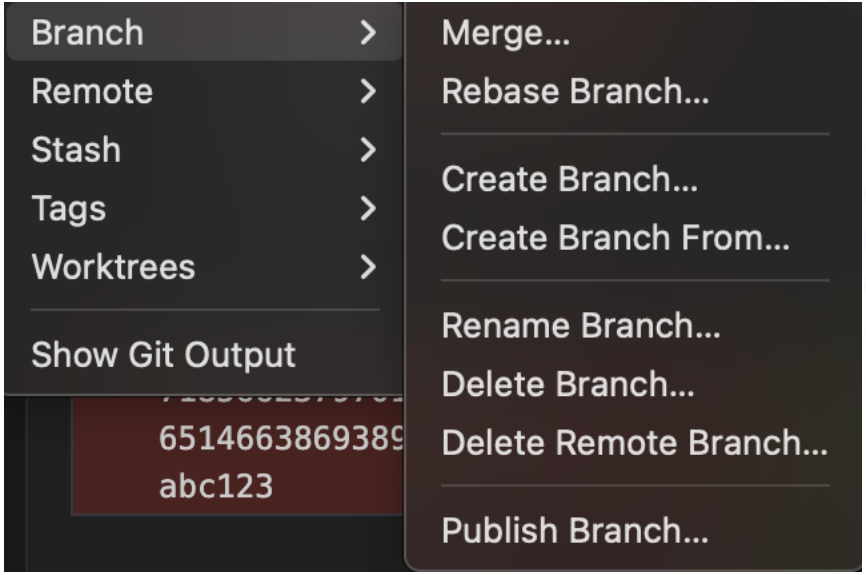
Polymorphism

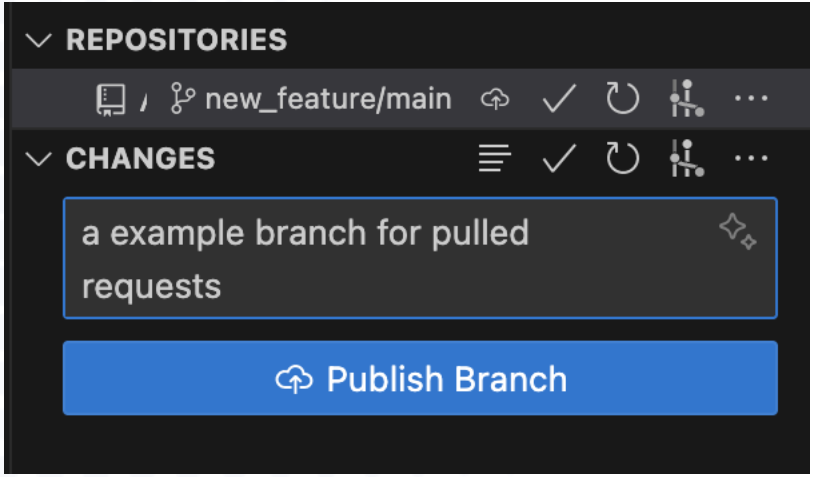
WORKING TOGETHER – DEFINE CLASSES WITHOUT CODE

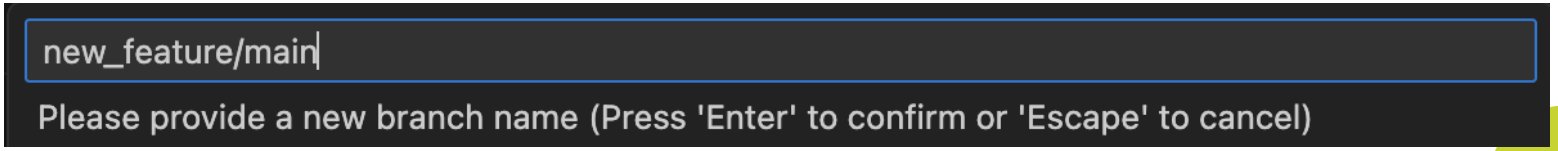

```
METHOD SIGNATURE:
    def NAME(param1: type, param2: type, …)-> return type
Brief description:
    What this function does.
```

WORKING TOGETHER – GIT PULLING REQUESTS

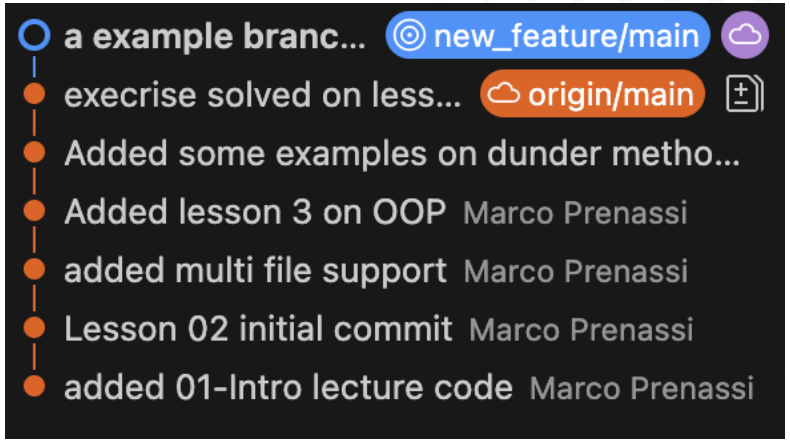
1. CREATE A NEW BRANCH FROM THE MAIN
CALL IT LIKE new_feature/category



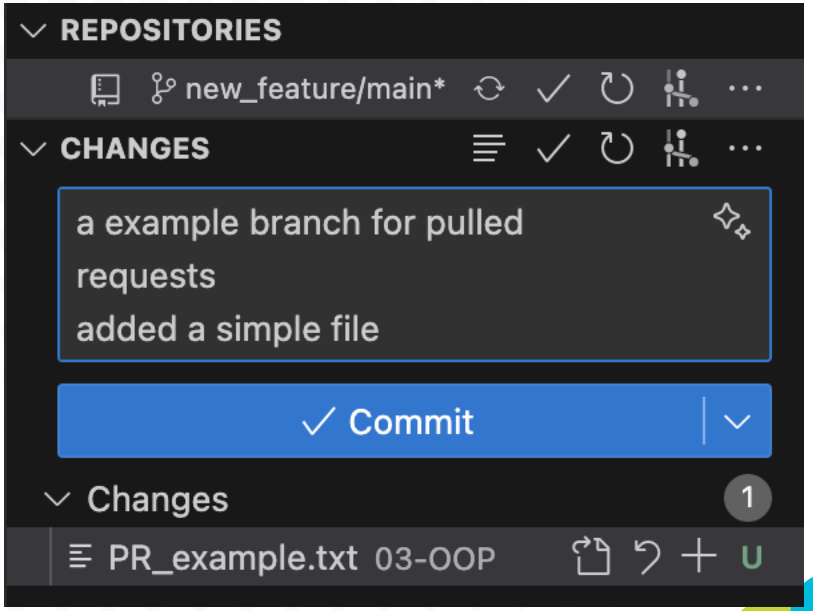




WORKING TOGETHER – GIT PULLING REQUESTS

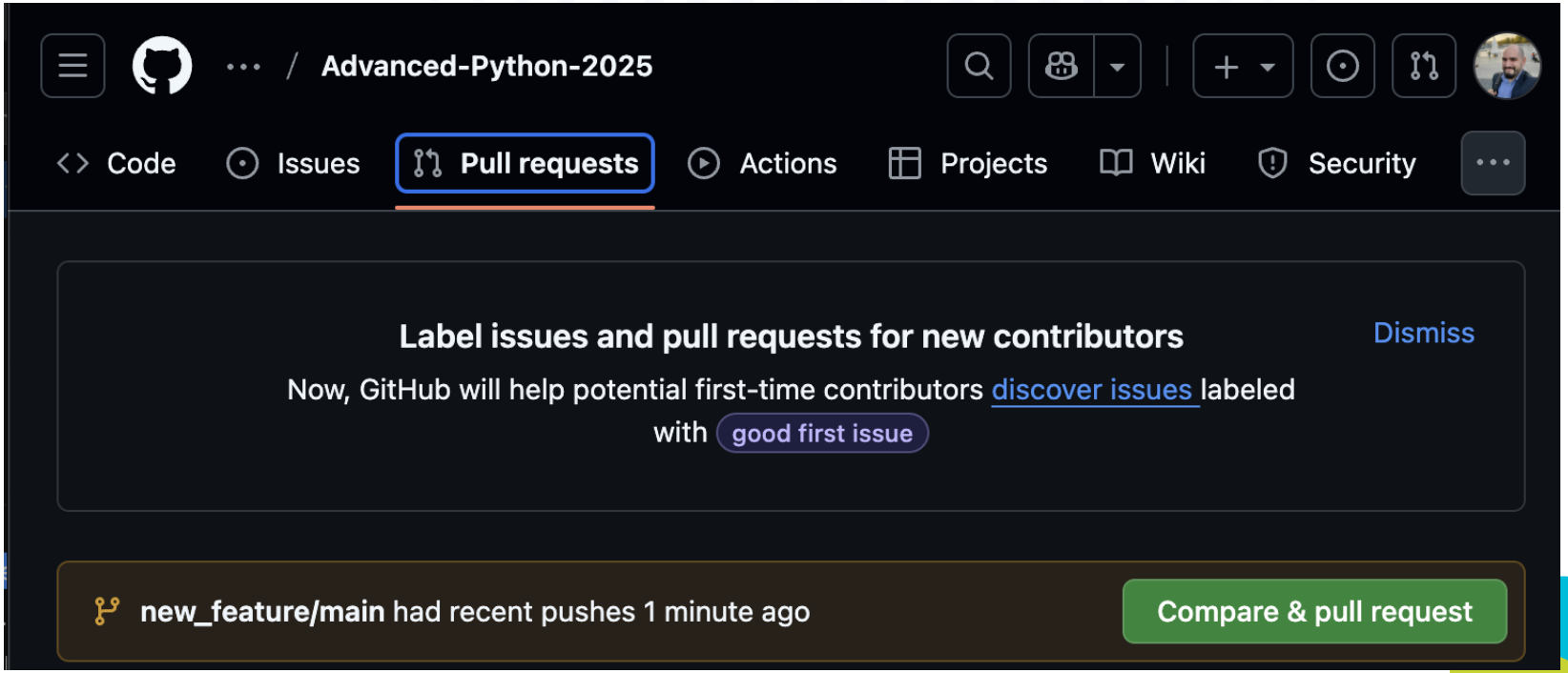


2. CHECKOUT IN THAT BRANCH, create/edit and TEST the new branch and then COMMIT and PUSH!



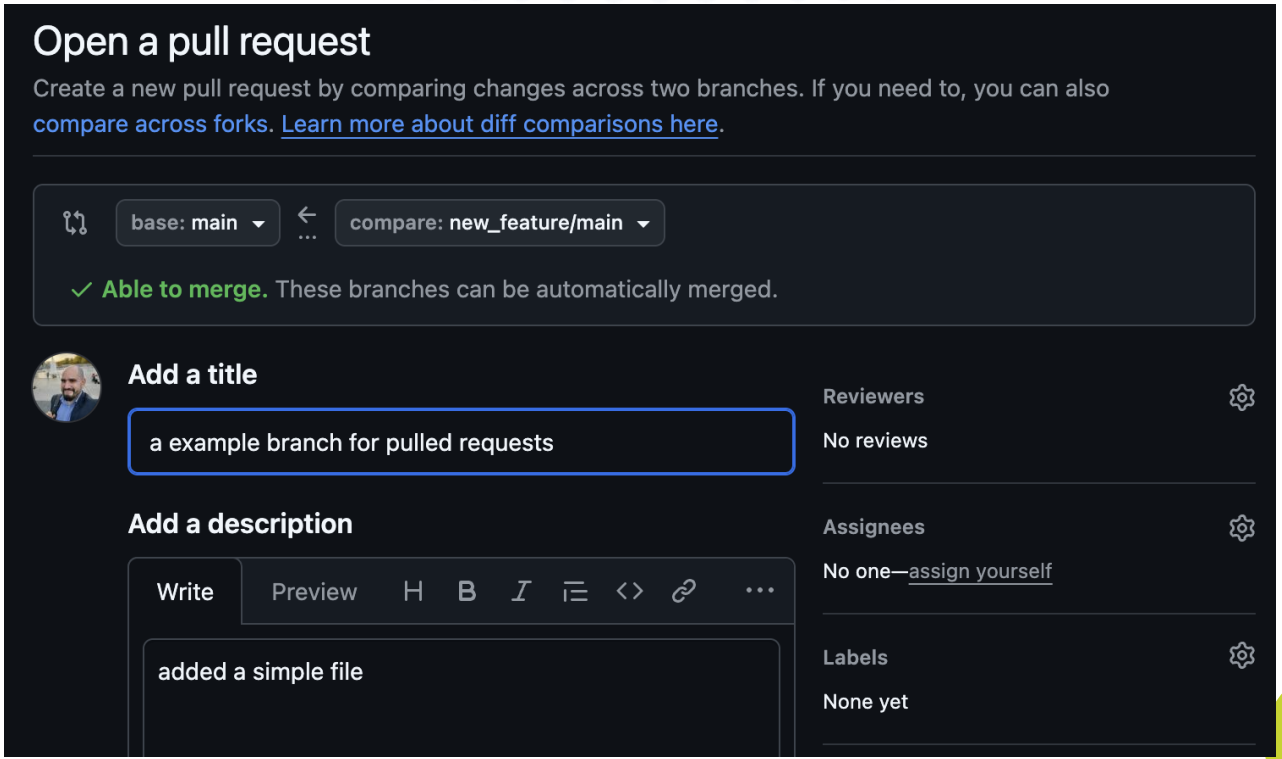
WORKING TOGETHER – GIT PULLING REQUESTS

2. In github you get a notification of the branch, you can do "Compare and send a Pull request"



WORKING TOGETHER – GIT PULLING REQUESTS

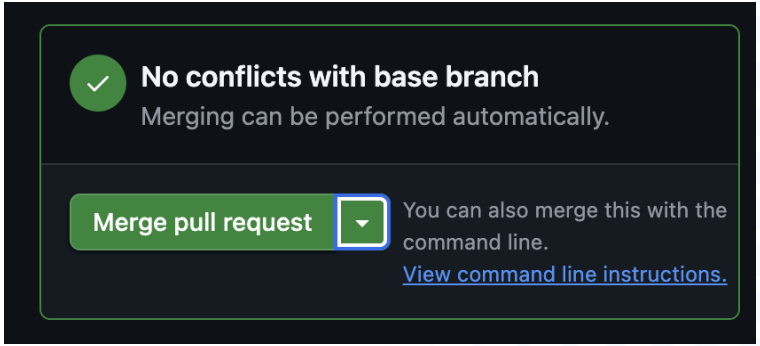
2. In github you get a new Pull request, compare it and check if there are some merge issue.



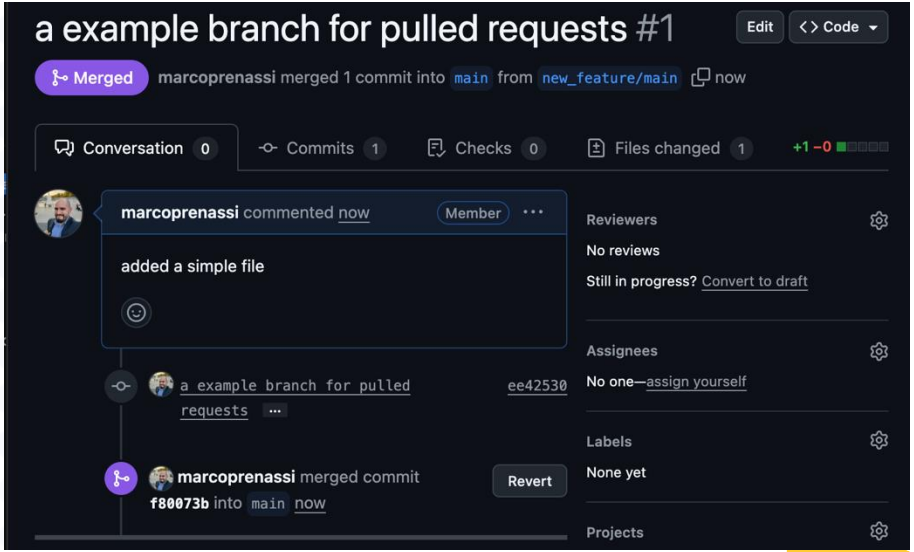
WORKING TOGETHER – GIT PULLING REQUESTS

3. The CODE MANTAINER (e.g., the team leader) check, approve and merge the Pull Request

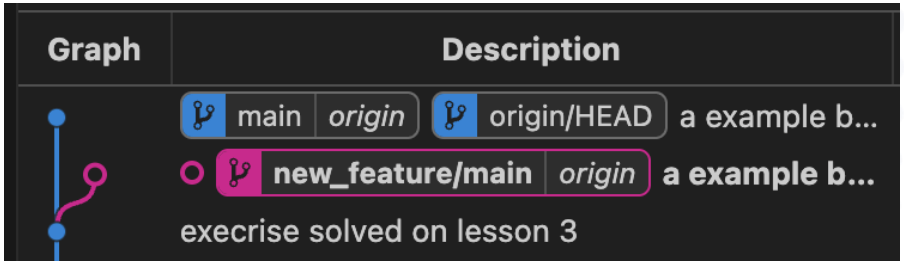
1



2



3



Finanziato dall'Unione europea
NextGenerationEU

Ministero dell'Università e della Ricerca

Italiadomani
PIANO NAZIONALE DI RIPRESA E RESILIENZA

nffa-di E-ARGO beanTech IT moves your business CERIC

Thank you

This Pilot training activity has been funded by the European Union – NextGenerationEU within the PNRR projects funded pursuant to Article 11, paragraph 1, of Notice 594/2024:

- "NFFA-DI cod. IR0000015, Missione 4, "Istruzione e Ricerca" – Componente 2, "Dalla ricerca all'impresa" – Linea di investimento 3.1, "Fondo per la realizzazione di un sistema integrato di infrastrutture di ricerca e innovazione" – Azione 3.1.1, "Creazione di nuove IR o potenziamento di quelle esistenti che concorrono agli obiettivi di Eccellenza Scientifica di Horizon Europe e costituzione di reti" (CUP B53C22004310006).
- "EFC cod. SSU2024-00002, Missione 4 "Istruzione e ricerca" - Componente 1, "Potenziamento dell'offerta dei servizi all'istruzione: dagli asili nido all'universita'" - Investimento 3.4 "Didattica e competenze universitarie avanzate" - Sub-Investimento "Rafforzamento delle scuole universitarie superiori" (CUP: G97G24000100007).