

Licenciatura en Sistemas

Materia: Introducción a la Programación

Docentes: Bagnes Patricia y Velcic Fernando

Integrantes: Galeano Carla y Gomez Sergio

Año: 2025



El presente informe tiene como objetivo documentar el desarrollo del trabajo práctico integrador para la materia Introducción a la Programación, cursada durante el año 2025 en la Universidad Nacional de General Sarmiento.

La consigna general del trabajo consistía en desarrollar una Pokédex web usando Python y Django, con el objetivo de que se vean bien presentados en una página de inicio y, además, que se permita buscar o filtrar los Pokemones según el tipo (agua, fuego, planta, etc). Todo esto lo hicimos aplicando los conocimientos adquiridos en clase, tales como: funciones, condicionales, estructuras básicas, y algunas herramientas adicionales externas como HTML y CSS.

Para comenzar, una de las primeras funciones que utilizamos para lograr traer las imágenes crudas desde la API fue la función **GetAllImages** de transport.py, que básicamente se encarga de obtener las imágenes como objetos JSON que necesitamos para armar las tarjetas de Pokémon, además de otros datos que irán en la tarjeta. Estas imágenes vienen “crudas”, es decir, sin procesar, y después las mostramos en el home del sitio web.

A continuación, nos trasladamos al layer services.py para mostrar el procedimiento de armado de tarjetas de Pokémon. En primer lugar definimos una lista vacía denominada “cartas”, luego, en una variable llamada “raw_image” importamos desde el transport.py las imágenes crudas desde la API. Luego, recorreremos mediante un ciclo cada imagen para convertirlas en tarjetas guardandolas en la variable llamada “carta”. Finalmente, agregamos cada card en la lista creada inicialmente y le pedimos a la función que las retorne.

```
# función que devuelve un listado de cards. Cada card representa una imagen de la API de Pokemon
def getAllImages():
    cartas = []
    # debe ejecutar los siguientes pasos:
    # 1) traer un listado de imágenes crudas desde la API (ver transport.py)
    # 2) convertir cada img. en una card.
    # 3) añadirlas a un nuevo listado que, finalmente, se retornará con todas las card encontradas.
    raw_image = transport.getAllImages()  ### 1) traemos todas las imágenes crudas llamando a la función getAllImages() de transport.py.
    for img in raw_image:                ### 2) en este ciclo, todas las imágenes crudas se transforman en cards.
        carta = translator.fromRequestIntoCard(img)
        cartas.append(carta)
    return cartas                        ### 3) se retorna un listado de cards y, además, con información de fácil acceso.
```

Continuando con la explicación, la función ya implementada **fromRequestIntoCard**, en el layer translator.py, fue la clave para transformar la información de cada Pokemon que viene desde la API como objetos JSON en algo útil para mostrar en la pantalla. Esto nos sirvió para armar las tarjetas de presentación de cada Pokemon.

En primer lugar, recibe como parámetro los datos crudos de un pokémon y los guarda en variables: ID (número del pokémon), name(nombre), height (altura), weight (peso) y base (experiencia base). También usa dos funciones del mismo layer, **safe_get** para obtener la imagen del pokémon de forma segura y **getTypes** para extraer sus tipos (fuego, agua, planta, etc).

Una vez que junta todo, devuelve la tarjeta lista para ser mostrada en la página web. Esta función nos ayudó a separar la lógica del “crudo” de la API del diseño de las tarjetas, haciendo el código más limpio.

```
# Usado cuando la información viene de la API, para transformarla en una Card.
def fromRequestIntoCard(poke_data):
    card = Card(
        id=poke_data.get('id'),
        name=poke_data.get('name'),
        height=poke_data.get('height'),
        weight=poke_data.get('weight'),
        base=poke_data.get('base_experience'),
        image=safe_get(poke_data, 'sprites', 'other', 'official-artwork', 'front_default'),
        types=getTypes(poke_data)
    )
    return card
```

El siguiente paso fue sumar el buscador que permite al usuario buscar un pokémon en específico. Utilizamos un condicional en donde evaluamos si lo que ingresa el usuario es distinto de una cadena vacía, en el caso de que lo sea, llamamos desde services.py a la función inicial **getAllImages** para guardar la información en la variable “allNames”. Iniciamos una lista vacía, recorremos con un ciclo for, y en el caso de que el nombre se encuentre dentro de la variable, la añadimos a la lista.

```
# función utilizada en el buscador.
def search(request):
    name = request.POST.get('query', '')

    # si el usuario ingresó algo en el buscador, se deben filtrar las imágenes por dicho ingreso.
    if (name != ''):
        allNames = services.getAllImages() ### se llama a getAllImages() para buscar informacion necesaria en la card.
        images = []
        for cards in allNames:
            if name == cards.name:          ### se busca si lo ingresado coincide con algun name en las cards.
                images.append(cards)
```

Un ejemplo de cómo se vería el home si el usuario ingresa la palabra “pikachu” en nuestro trabajo:



Además, en el layer `services.py`, en caso de que el usuario quiera filtrar los pokemones según su tipo, desarrollamos la siguiente función llamada **filteredByType**.

En primer lugar creamos una lista vacía, luego usamos un ciclo para recorrer cada tarjeta dentro de la información obtenida de **getAllImages**. Mediante un condicional evaluamos si el tipo de pokemon coincide con la información, en caso de que lo sea, la añadimos a la lista vacía y la retornamos.

```
# función que filtra las cards según su tipo.
def filteredByType(type_filter):
    filtered_cards = []

    for card in getAllImages(): ### se trabaja con el listado de cards y se busca una información específica.
        # debe verificar si la casa de la card coincide con la recibida por parámetro. Si es así, se añade al listado de filtered_cards.
        if type_filter in card.types:
            filtered_cards.append(card)

    return filtered_cards
```

La forma en la que logramos mostrar la información en el home, es de la siguiente manera: en views.py, utilizamos la función llamada **filter_by_types(request)**, en donde extraemos la información que previamente filtramos.

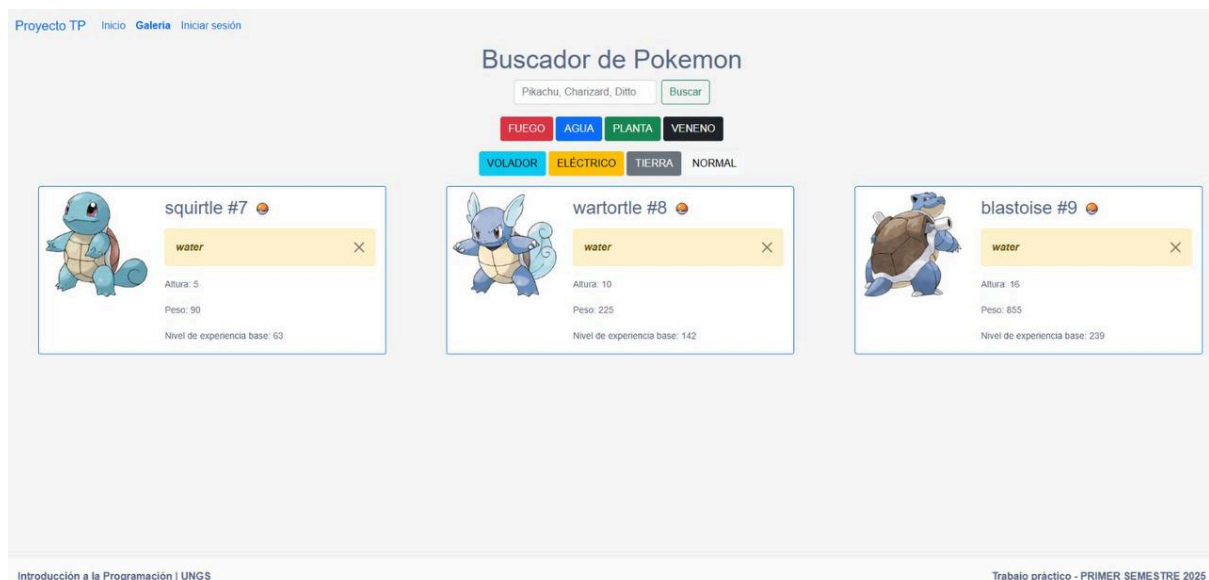
```
# función utilizada para filtrar por el tipo del Pokemon
def filter_by_type(request):
    type = request.POST.get('type', '')

    if type != '':
        images = services.filterByType(type) # debe traer un listado filtrado de imágenes, segun si es o contiene ese tipo.
        favourite_list = []

        return render(request, 'home.html', { 'images': images, 'favourite_list': favourite_list })
    else:
        return redirect('home')
```

Lo que hace esta función es evaluar mediante un condicional si lo que ingresa el usuario no es una cadena vacía para traernos el listado ya filtrado de imágenes. Luego las muestra en el home.

Quedaría algo como esto:

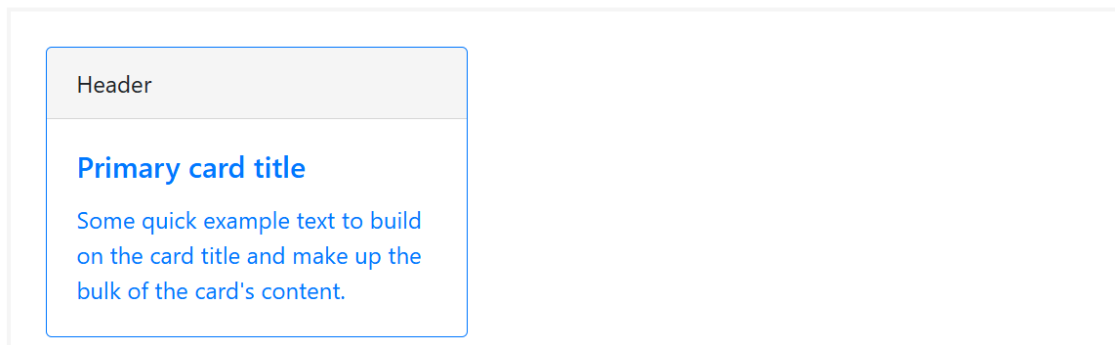


Por último, obligatoriamente debíamos lograr que los bordes de las tarjetas sean de determinados colores según el tipo de pokémon. Esto lo logramos utilizando condicionales y sobre todo nos sirvió buscar información sobre la sintaxis de HTML que no trabajamos en clase.

En primer lugar, hicimos uso de la información de “Bootstrap” que se nos proporcionaba en el repositorio base. Para armar los bordes con distintos colores se nos mostraba de la siguiente manera:

Border

Use [border utilities](#) to change just the `border-color` of a card. Note that you can put `.text-{color}` classes on the parent `.card` or a subset of the card's contents as shown below.



```
<div class="card border-primary mb-3" style="max-width: 18rem;">
  <div class="card-header">Header</div>
  <div class="card-body text-primary">
    <h5 class="card-title">Primary card title</h5>
    <p class="card-text">Some quick example text to build on the card title and make up the b
  </div>
```

Copy

En segundo lugar, desarrollamos condicionales que nos permita corroborar por cada tipo de Pokémon. Y por último, buscamos como usar condicionales dentro de un “DIV”. Para estos, miramos los mismos que ya estaban incorporados en el código.

El resultado es el siguiente:

- **En código**

```










<div class="col">
  <!-- evaluar si la imagen pertenece al tipo fuego, agua o planta -->
  <div class="card"
    {%if 'fire' in img.types %}
      border-danger
    {%elif 'water' in img.types %}
      border-primary
    {%elif 'grass' in img.types %}
      border-success
    {% else %}
      border-warning
    {% endif %}
    mb-3 ms-5" style="max-width: 540px;">

    <div class="row g-0">
      <div class="col-md-4">
        
      </div>

      <div class="col-md-8">
        <div class="card-body">
          <h3 class="card-title">{{ img.name }} #{{ img.id }} 

```

- En página

 <p>bulbasaur #1 🌿</p> <p>grass poison ✕</p> <p>Altura: 7 Peso: 69 Nivel de experiencia base: 64</p>	 <p>ivysaur #2 🌿</p> <p>grass poison ✕</p> <p>Altura: 10 Peso: 130 Nivel de experiencia base: 142</p>	 <p>venusaur #3 🌿</p> <p>grass poison ✕</p> <p>Altura: 20 Peso: 1000 Nivel de experiencia base: 236</p>
 <p>charmander #4 🔥</p> <p>fire ✕</p> <p>Altura: 6 Peso: 85 Nivel de experiencia base: 62</p>	 <p>charmeleon #5 🔥</p> <p>fire ✕</p> <p>Altura: 11 Peso: 190 Nivel de experiencia base: 142</p>	 <p>charizard #6 🔥</p> <p>fire flying ✕</p> <p>Altura: 17 Peso: 905 Nivel de experiencia base: 240</p>
 <p>squirtle #7 💧</p> <p>water ✕</p> <p>Altura: 5 Peso: 90 Nivel de experiencia base: 63</p>	 <p>wartortle #8 💧</p> <p>water ✕</p> <p>Altura: 10 Peso: 225 Nivel de experiencia base: 142</p>	 <p>blastoise #9 💧</p> <p>water ✕</p> <p>Altura: 16 Peso: 855 Nivel de experiencia base: 239</p>

Introducción a la Programación | UNGS

Trabajo práctico - PRIMER SEMESTRE 2025

Para concluir este informe, el trabajo práctico de IP nos aporta mucha información nueva con la que podemos relacionarnos. Todo lo trabajado en clases como variables, ciclos, condicionales, funciones etc, se ven en el código de una forma más desarrollada. A pesar de parecer algo ajeno a lo que estudiamos durante el semestre, al final se trata de Python y algunas cosas nuevas (Django o DIV en HTML). Con esto, comprendemos que podemos trabajar Python a un nivel más allá de realizar actividades matemáticas y llevarlo a un ámbito más profesional.