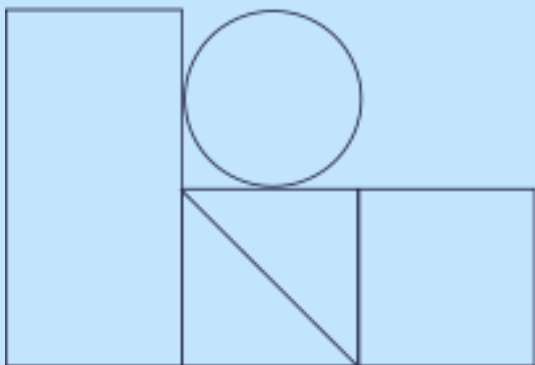


Django

Configurando Django



Índice

Instalación y configuración de Python	3
Instalación de Django	9
Creación de un proyecto Django	11
Parámetros en la URL	19

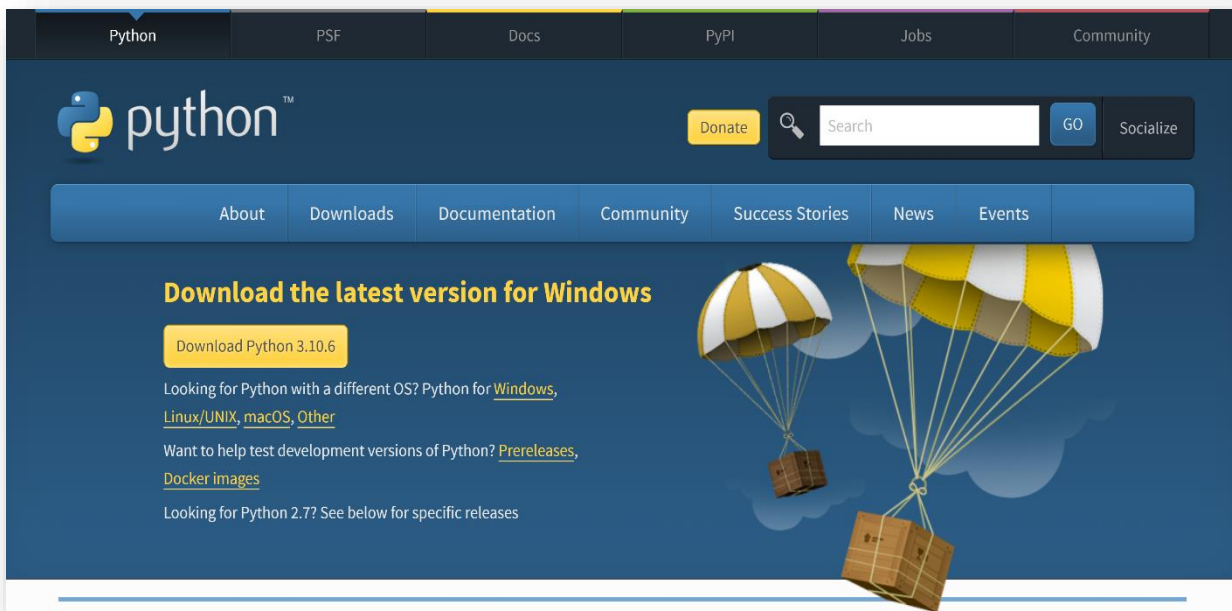
Instalación y configuración de Python

Para poder trabajar con Python, lo primero que necesitaremos hacer será instalar el propio **Python** en el sistema y agregarlo al **path**.

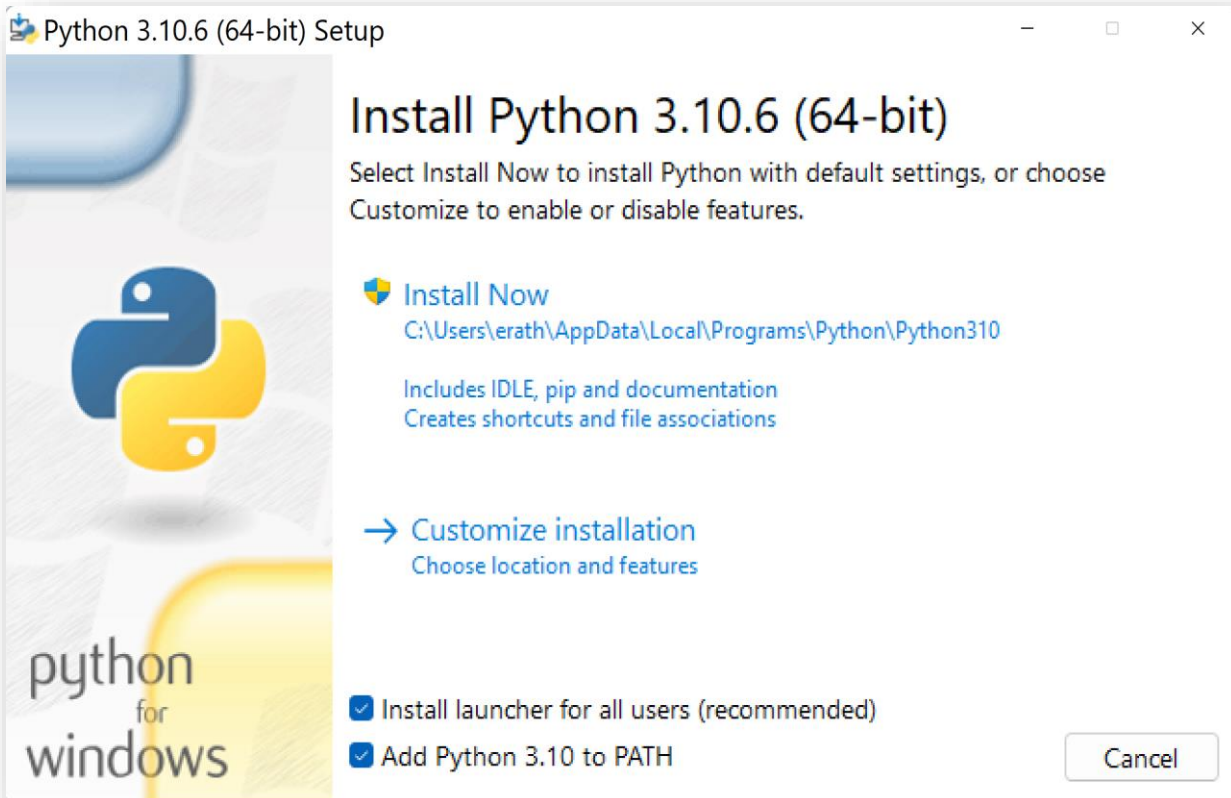
Para ello nos bajamos la última versión disponible en:

<https://www.python.org/downloads/>

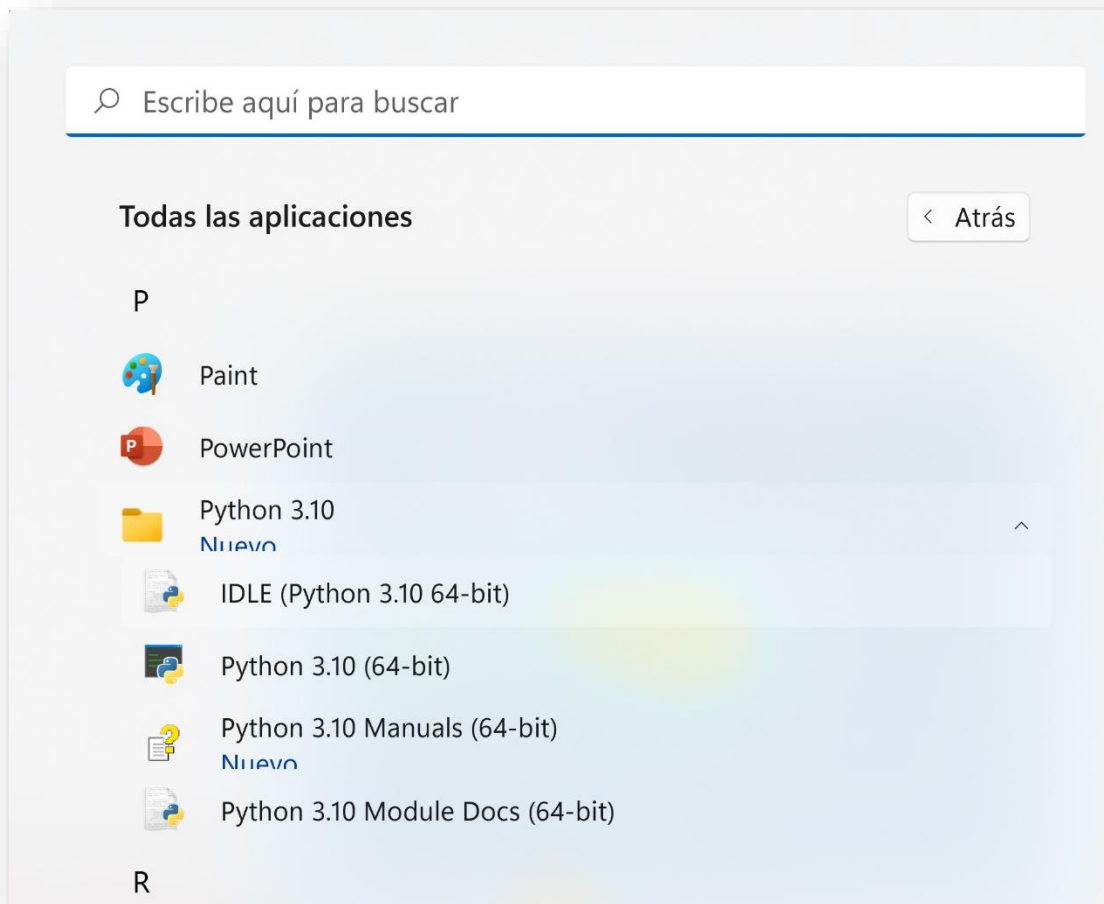
En el momento de la confección de este curso es la V3.10.6:



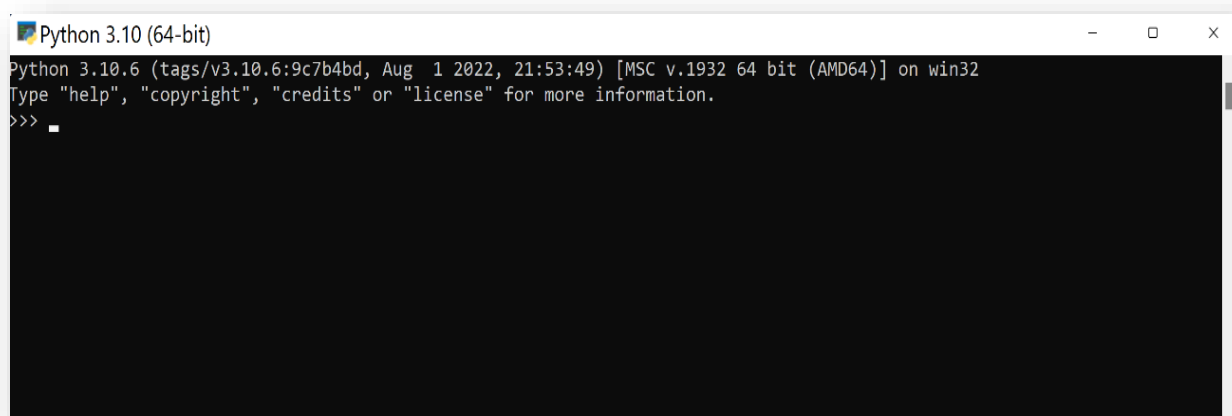
La instalación es muy sencilla y lo único que tenemos que tener en cuenta es asegurarnos de añadir Python al **Path** de nuestro sistema operativo. Para ello marcaremos la casilla correspondiente. El resto de opciones las dejaremos por defecto:



Una vez instalado Python en nuestro sistema, dispondremos del Shell de Python con el que podremos comenzar a programar directamente:



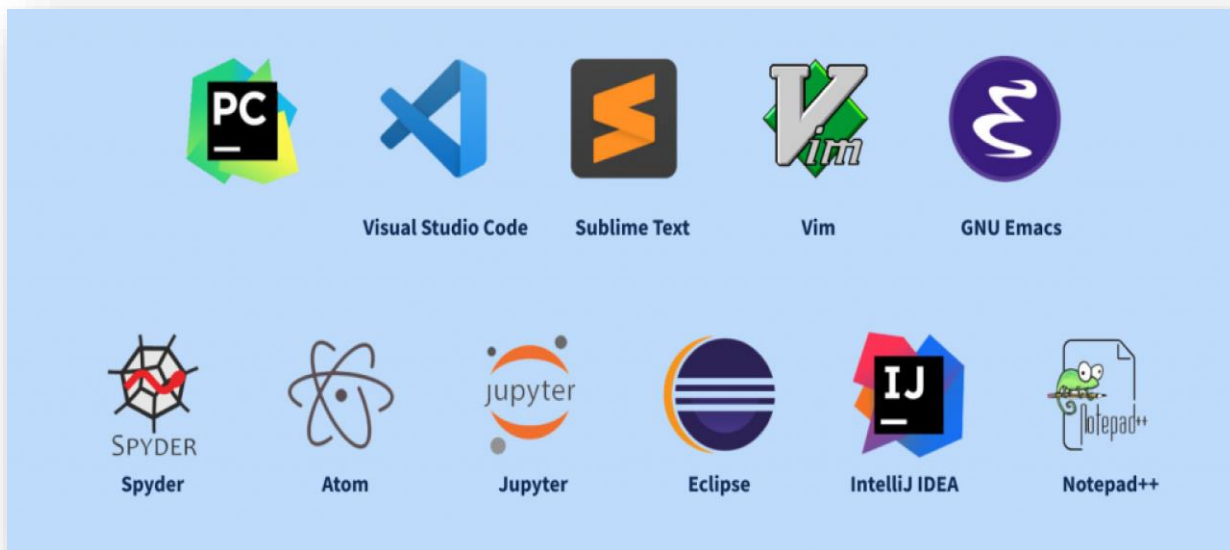
No es más que una ventana de comandos lista para recibir órdenes en **Python**:



No obstante, resulta incómodo trabajar con ella, ya que no nos brinda ningún tipo de ayuda, por lo que lo siguiente que necesitaremos es un **IDE** (entorno de desarrollo integrado) con el que trabajar.

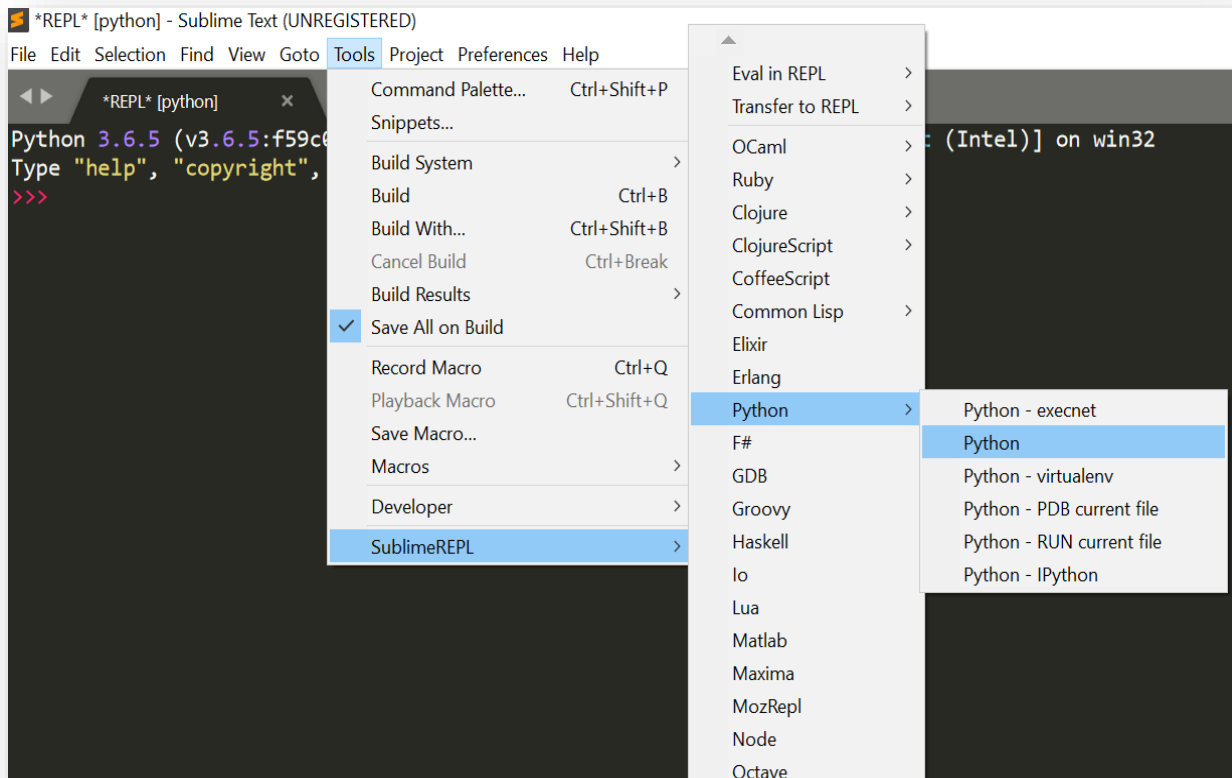
Para programar en **Python** se pueden usar numerosos **IDEs**. La mejor opción es, sin duda instalar Anaconda, un framework de Python que nos instala el lenguaje y una extensa cantidad de herramientas y recursos. También tenemos opciones como **Visual Studio Code** con el plug-in de **Python** o el **Pycharm** de la casa **JetBrains**. Este último goza de una inmejorable reputación y, aunque es de pago, se ofrece una versión gratuita bastante completa. Otra buena opción es **Sublime Text** y **Eclipse** con el plug-in **PYDev**, ambos gratuitos.

Respecto al tema de qué **IDE** es mejor para programar en **Python**, la elección es totalmente personal. Cada programador tiene sus preferencias, pero, en general, todos tienen más o menos las mismas capacidades y funcionalidades.



Si usamos, por ejemplo, **Sublime Text** tenemos que configurarlo para programar en **Python**, para ello iremos a la opción;

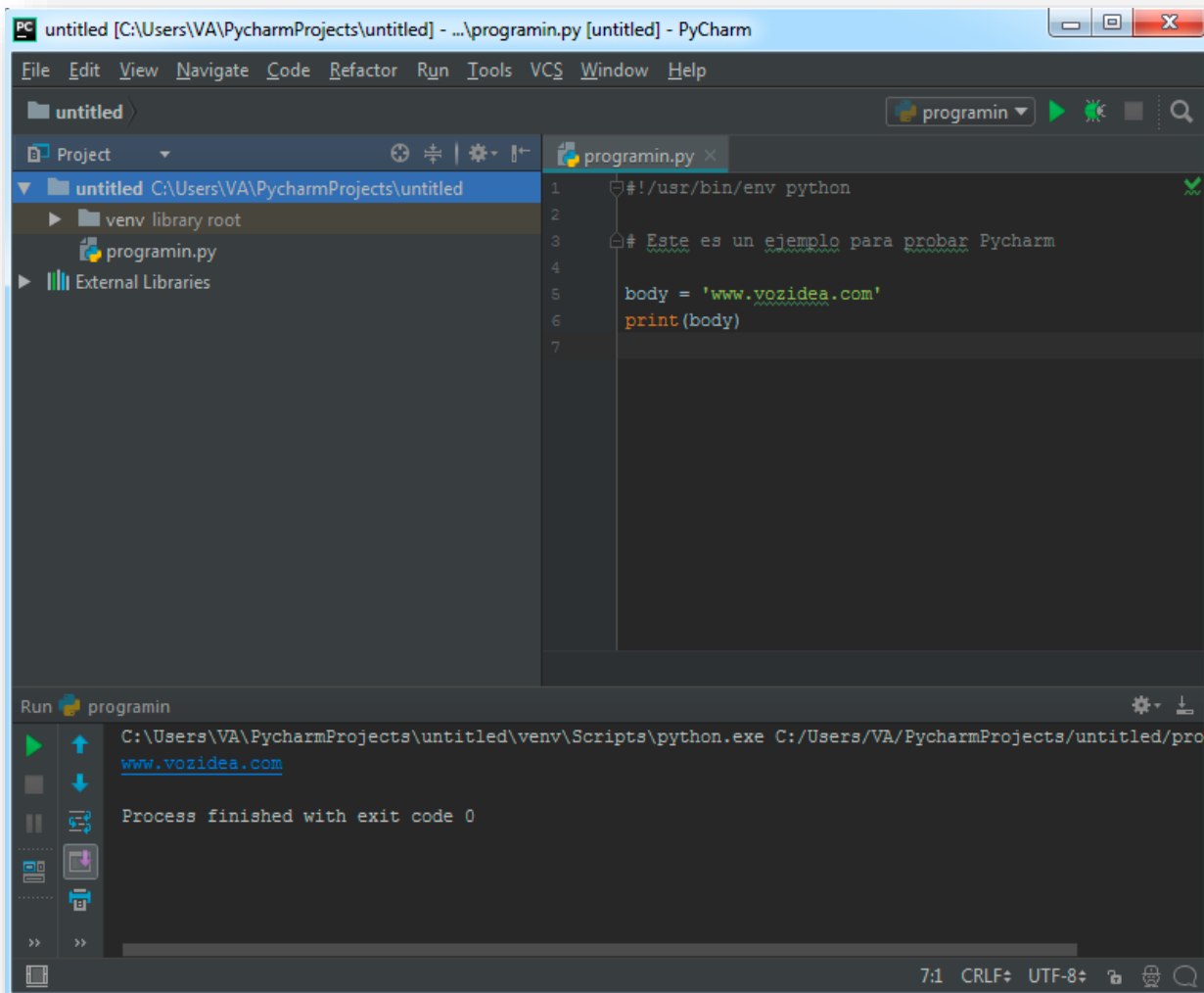
Tools/Command Pallette y dentro de esta, **Install Package Control**, luego usar el comando **Package Control:Install Package**. Finalmente **SublimeREPL** y listo, ya podemos empezar a programar en **Python**:



En el caso de **Visual Studio Code** deberemos instalar el plug-in **Python**:

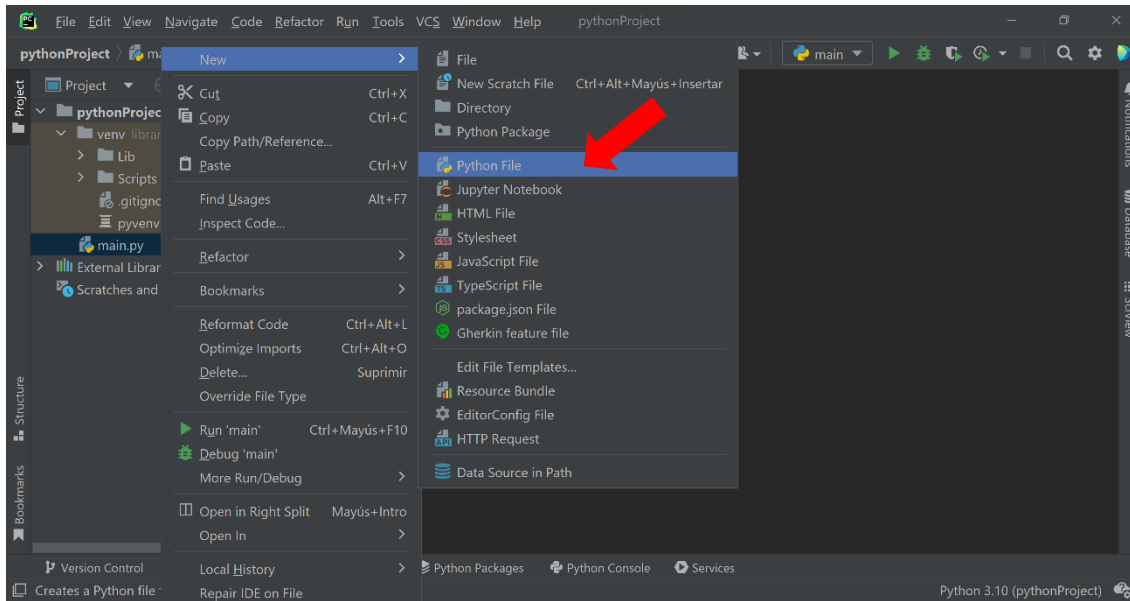


En el caso de **Pycharm** ya viene con todo lo necesario para empezar a trabajar con Python.



Para el resto de **IDEs** disponibles en el mercado cada uno dispone de documentación de referencia en su respectiva página web sobre instalación, plug-ins, etc.

Sea cual sea nuestra elección, una vez instalado **Python** y nuestro **IDE** lo único que deberemos hacer es crear un archivo **Python** (con la extensión `.py`), darle un nombre y ya podremos trabajar sobre él:

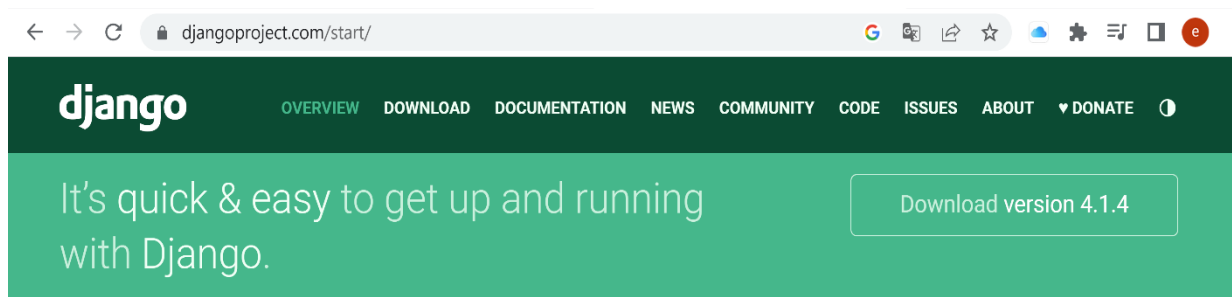


Instalación de Django

En MAC y en LINUX ya viene instalado Django.

En Windows necesitamos primero tener instalado Python.

Lo bajamos desde la página oficial:



Getting started with Django

Depending how new you are to Django, you can [try a tutorial](#), or just [dive into the documentation](#).

Want to learn more about Django? Read the overview to see whether Django is right for your project.

Si tenemos Python instalado podemos instalar Django con el PIP.



```
Administrador: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Instale la versión más reciente de PowerShell para obtener nuevas características y mejoras. https://aka.ms/PSWindows

PS C:\WINDOWS\system32> python --version
Python 3.11.1
PS C:\WINDOWS\system32> pip install Django==4.1.4
Collecting Django==4.1.4
  Downloading Django-4.1.4-py3-none-any.whl (8.1 MB)
    ----- 8.1/8.1 MB 18.5 MB/s eta 0:00:00
Collecting asgiref<4,>=3.5.2
  Downloading asgiref-3.6.0-py3-none-any.whl (23 kB)
Collecting sqlparse>=0.2.2
  Downloading sqlparse-0.4.3-py3-none-any.whl (42 kB)
    ----- 42.8/42.8 kB ? eta 0:00:00
Collecting tzdata
  Downloading tzdata-2022.7-py2.py3-none-any.whl (340 kB)
    ----- 340.1/340.1 kB 22.0 MB/s eta 0:00:00
Installing collected packages: tzdata, sqlparse, asgiref, Django
Successfully installed Django-4.1.4 asgiref-3.6.0 sqlparse-0.4.3 tzdata-2022.7
PS C:\WINDOWS\system32>
```

Para comprobar que se ha instalado todo correctamente nos metemos en el intérprete de Python:

```
PS C:\WINDOWS\system32> python
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec 6 2022, 19:58:39) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import django
>>> django.VERSION
(4, 1, 4, 'final', 0)
```

Creación de un proyecto Django

Django se puede instalar en local o en virtual, se suele instalar en virtual ya que este tipo de instalación presenta numerosas ventajas.

Si tenemos Django instalado en local, nuestras versiones Django y de Python serán las que se utilicen en todos nuestros proyectos. Y de la misma forma las dependencias que tengamos instaladas serán comunes a todos los proyectos.

Si instalamos Django en un entorno virtual, tendremos la posibilidad de forzar a que todos los programadores del proyecto trabajen con la misma versión, tanto de Python como de Django, lo cual ahorra muchos problemas de incompatibilidades.

Además, usando un entorno virtual se tiene mucho más control sobre los entornos de programación de nuestro proyecto.

Lo normal es que un proyecto tenga un entorno de “desarrollo” y una de “producción”, e incluso muchas veces un tercero intermedio de “pruebas”.

En nuestro entorno virtual podremos hacer todas las pruebas y modificaciones que sean necesarias en “desarrollo” antes de aunar los cambios y pasarlos al entorno definitivo de “producción”. Y en dichas pruebas podremos desplegar nuestro programa en una o varias versiones de Django y Python.

Podemos, además, centralizar las diferentes dependencias que van a necesitar nuestros equipos de programadores y, como hemos visto, igualar los entornos de desarrollo-pruebas-producción.

Para nuestro primer proyecto, en principio vamos a trabajar con Django en local.

Llegado este punto, es interesante hacer una aclaración sobre Django y las BBDD.

Los gestores de bases de datos soportados oficialmente por Django son:

- **SQLite3:** Gestor de BBDD por defecto.
- **PostgreSQL:** Gestor recomendado.
- **MySQL**
- **Oracle**

El gestor de BBDD por defecto es SQLite3. Es un gestor muy ligero que viene ya incorporado en las últimas versiones de Python, por lo que es la mejor opción para empezar a trabajar ya que no necesitaremos instalar ni configurar nada.

El gestor recomendado por Django es PostgreSQL.

Para sistemas gestores de bases de datos que no están por defecto incluidos en Django (como SQL server, SAP SQL, etc) existen en internet “conectores” que permiten a Django trabajar con prácticamente cualquier BBDD.

Algunos conectores ofrecidos por terceros son:

- **SQL Server**
- **SAP SQL**
- **Firebird**
- **Etc.**

Una vez aclarado este punto, podemos comenzar con nuestro primer proyecto Django en local.

Para crear nuestro proyecto Django deberemos situarnos en la carpeta de trabajo de dicho proyecto (la que nosotros elijamos, no hay restricciones) y ejecutar el siguiente comando:

django-admin startproject miProyecto

Donde “**miProyecto**” será el nombre de nuestra app. Con lo que se nos creará la carpeta con los archivos necesarios para el proyecto.

Nombre	Tamaño	Puntuación	Tipo	Modificación	Ancho
miProyecto			Archivo carpeta	Hoy 14:08	
manage.py	688 bytes		Python File	Hoy 14:08	

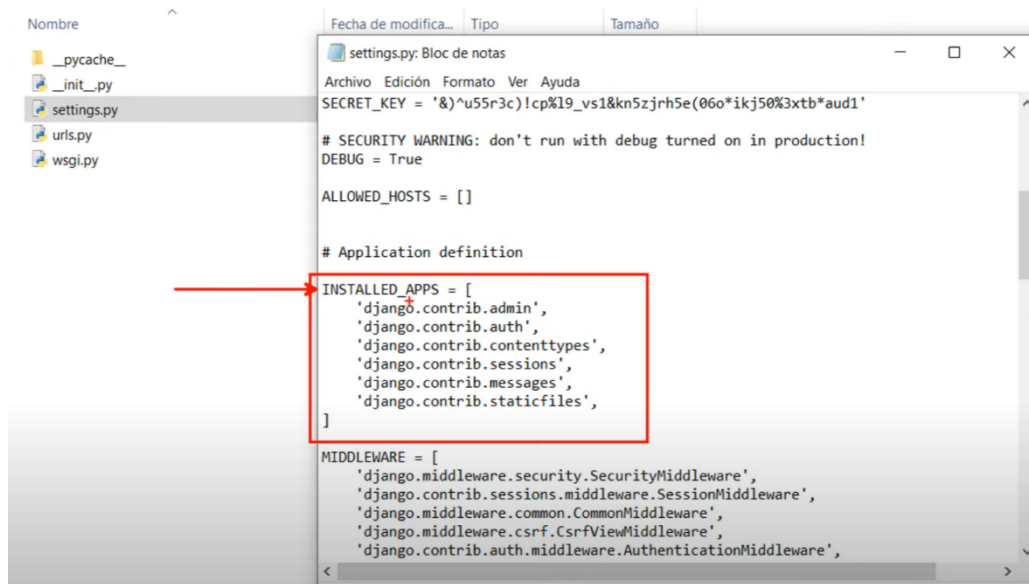
Veremos que nos ha creado un archivo llamado **manage.py**. Este archivo es muy importante ya que es una utilidad de línea de comandos que nos permite interactuar con nuestros proyectos Django de varias formas.

Nombre	Tamaño	Puntuación	Tipo	Modificación	Ancho
init.py	0 bytes		Python File	Hoy 14:08	
asgi.py	413 bytes		Python File	Hoy 14:08	
settings.py	3,27 KB		Python File	Hoy 14:08	
urls.py	773 bytes		Python File	Hoy 14:08	
wsgi.py	413 bytes		Python File	Hoy 14:08	

El archivo **_init_.py** es un archivo necesario para que Python trate el directorio de mi sitio como si fuese un paquete.

El archivo **settings.py** contiene todas las configuraciones de nuestro proyecto de Django.

Contiene entre otras cosas las aplicaciones que vienen instaladas con Django.



El archivo `urls.py` es el que almacena las URLs que va a utilizar nuestro proyecto.

`wsgi.py` es un archivo relativo al servidor WEB que vamos a utilizar.

Para que todas estas aplicaciones funcionen, debemos instalar nuestra BBDD.

Por defecto la BBDD que viene con Django es SQLite.

En la carpeta de nuestro proyecto usaremos el siguiente comando:

`python manage.py migrate`

```
Administrador: Windows PowerShell
PS G:\WORKSPACE\11 DJANGO\miProyecto> python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK
PS G:\WORKSPACE\11 DJANGO\miProyecto>
```

Podremos observar que en nuestra carpeta de trabajo aparece la BBDD:

Nombre	Tamaño	Puntuación	Tipo	Modificación
miProyecto			Archivo carpeta	Hoy 14:08
db.sqlite3	128 KB		Archivo SQLITE3	Hoy 18:51
manage.py	688 bytes		Python File	Hoy 14:08

Con esto ya tendríamos nuestro proyecto en funcionamiento.

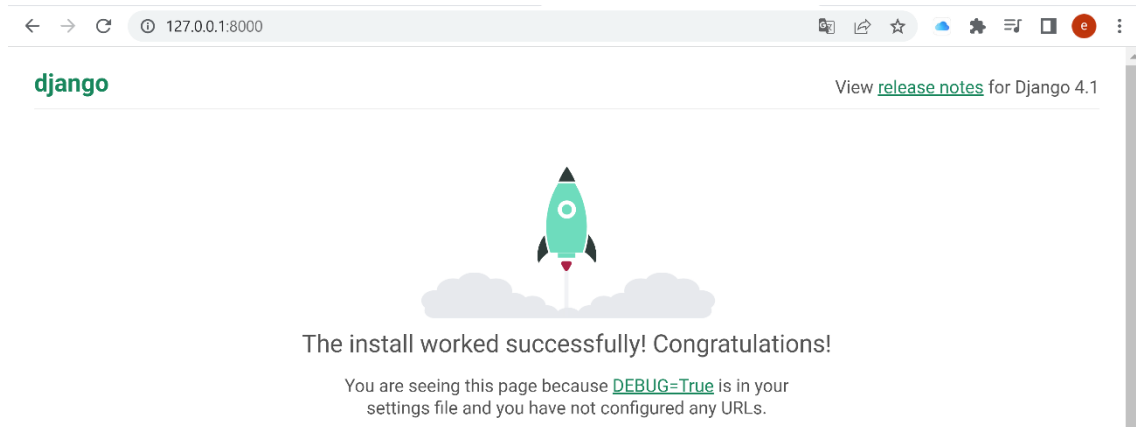
Para comprobarlo ejecutaríamos el comando:

Python manage.py runserver

```
Administrador: Windows PowerShell
PS G:\WORKSPACE\11 DJANGO\miProyecto> python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
December 27, 2022 - 18:55:22
Django version 4.1.4, using settings 'miProyecto.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Si abrimos esa dirección en nuestro navegador veremos nuestro proyecto en funcionamiento:



Nota: En el caso de que estemos usando Visual Studio Code podemos introducir todos los comandos que hemos visto en la consola de comandos que viene incluida en el IDE.

Ahora que ya tenemos el proyecto funcionando, vamos a crear nuestra primera página personalizada.

Anteriormente vimos la forma de trabajar de Django y su filosofía de **template-view-model**. Para nuestra primera página no vamos a necesitar conectar a BBDD ni crear ningún **template**.

Lo que sí que vamos a necesitar es el **view**, ya que vamos a realizar una petición al servidor, le vamos a pedir que muestre una URL.

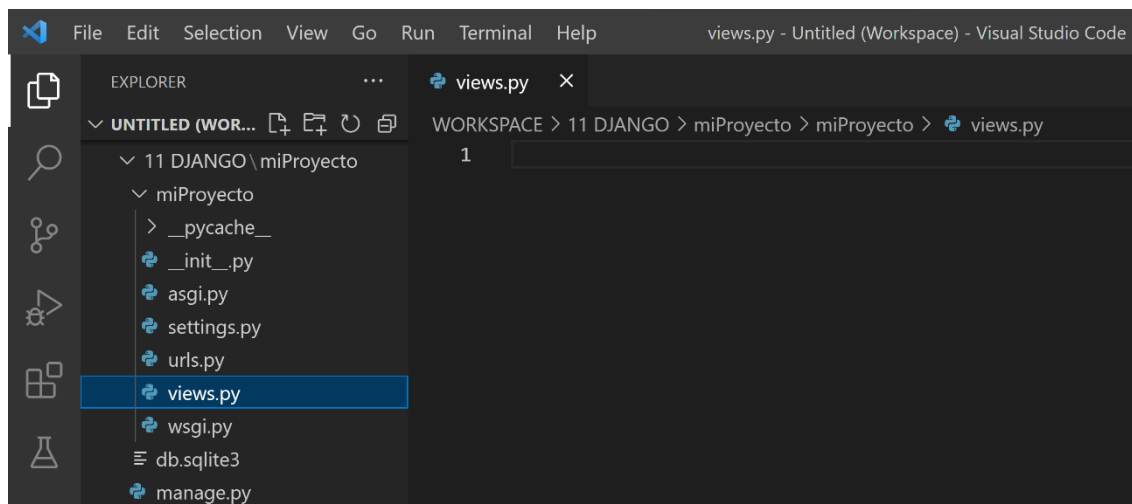
Esa petición la recibe el **view** que vamos a crear y luego nos enviará la información.

Para hacer esto, vamos a trabajar con objetos de dos clases de la biblioteca de Django:

- **Request**: Para hacer la petición.
- **HttpResponse**: Para enviar la información.

Cuando se solicita una página, Django crea un objeto **HttpRequest** que contiene metadatos sobre la solicitud. Luego, Django carga la vista apropiada pasando **HttpRequest** como primer argumento a la función. Cada vista es responsable de devolver un objeto **HttpResponse**.

Lo primero que haremos es crear un archivo que corresponderá a las vistas que vayamos creando. Por convención se le suele llamar **views.py**.



En este punto, al crear el archivo con extensión **.py**, si tenemos Visual Studio Code recién instalado, el programa detectará que estamos trabajando con Python y nos sugerirá instalar el plug-in de dicho lenguaje:



Procedemos a la instalación y ya podremos seguir trabajando con Python en Visual Studio.

Dado que, necesitaremos usar los objetos **Request** y **HttpResponse** deberemos importarlos desde la biblioteca de Django:

```
from django.http import HttpResponse
```

Ahora debemos declarar nuestra vista. Para ello simplemente creamos una función que va a recibir por parámetro un **request**.

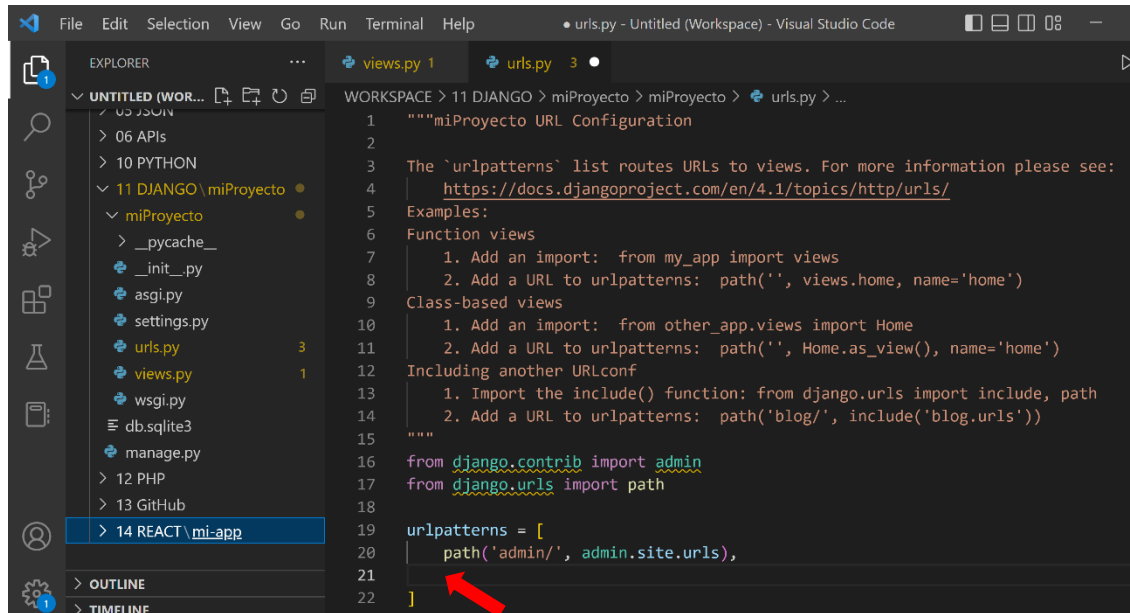
A cada función que creamos en el archivo **views.py** se le llama vista.

En nuestro caso vamos a crear una función sencilla que saque un mensaje por pantalla:

```
def saludo(request):  
    return HttpResponse("Hola mundo")
```


Acabamos de crear nuestra primera vista.

Ahora debemos decirle a Python cuál es la URL que debemos introducir en el navegador para ir a esta vista. Eso lo haremos en el archivo **urls.py**:



The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays the project structure under '11 DJANGO \miProyecto'. The 'urls.py' file is highlighted. The main editor window shows the content of 'urls.py', which includes comments about URL configuration, examples of function and class-based views, and the 'urlpatterns' list. A red arrow points to the 'urlpatterns' list, specifically to the 'path' function call.

```
1 """miProyecto URL Configuration
2
3 The 'urlpatterns' list routes URLs to views. For more information please see:
4 |     https://docs.djangoproject.com/en/4.1/topics/http/urls/
5 Examples:
6 Function views
7     1. Add an import: from my_app import views
8     2. Add a URL to urlpatterns: path('', views.home, name='home')
9 Class-based views
10    1. Add an import: from other_app.views import Home
11    2. Add a URL to urlpatterns: path('', Home.as_view(), name='home')
12 Including another URLconf
13    1. Import the include() function: from django.urls import include, path
14    2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))
15 """
16 from django.contrib import admin
17 from django.urls import path
18
19 urlpatterns = [
20     path('admin/', admin.site.urls),
21 ]
```

En este archivo encontraremos una lista que contienen el **path** de todas las vistas. Por defecto ya contiene una, **admin**.

Añadimos nuestra vista dándole un nombre (que no necesariamente tiene que ser el mismo de la vista), seguido de una coma y el nombre de la vista.

Deberemos también importar la vista:

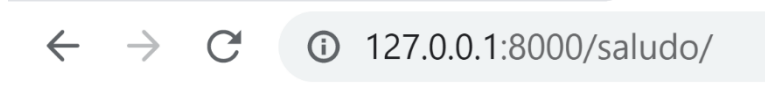
```

3 The `urlpatterns` list routes URLs to views. For more information please see:
4   https://docs.djangoproject.com/en/4.1/topics/http/urls/
5 Examples:
6 Function views
7     1. Add an import: from my_app import views
8     2. Add a URL to urlpatterns: path('', views.home, name='home')
9 Class-based views
10    1. Add an import: from other_app.views import Home
11    2. Add a URL to urlpatterns: path('', Home.as_view(), name='home')
12 Including another URLconf
13    1. Import the include() function: from django.urls import include, path
14    2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))
15 """
16 from django.contrib import admin
17 from django.urls import path
18 from miProyecto.views import saludo
19
20 urlpatterns = [
21     path('admin/', admin.site.urls),
22     path('saludo', saludo)
23 ]
24

```

Ya tenemos nuestra vista creada y registrada.

Podemos levantar el servidor y ver en el navegador el resultado:



Hola mundo

Podemos poner incluso estilos a los elementos:

```

from django.http import HttpResponse

#Definición de la vista:
def saludo(request):
    return
    HttpResponse("<html><body><h1>Hola
    mundo</h1></body></html>")

```

En estos casos se suele meter el texto en una variable:

```
from django.http import HttpResponse

#Definición de la vista:
def saludo(request):
    texto = "<html><body><h1>Hola mundo</h1></body></html>"
    return HttpResponse(texto)
```

Podemos ponerlo de forma más elegante:

```
from django.http import HttpResponse

#Definición de la vista:
def saludo(request):
    texto = """
    <html>
    <body>
    <h1>Hola mundo</h1>
    </body>
    </html>
    """
    return HttpResponse(texto)
```



Parámetros en la URL

Es muy frecuente cuando se trabaja con sitios WEB dinámicos tener la necesidad de pasar parámetros a la URL. Vamos a ver cómo manejar contenido dinámico y pasar parámetros de la URL con Django.

Vamos a realizar un script que muestre la hora y día del sistema.

Para crear contenido dinámico:

Creamos una nueva vista:

```
from django.http import HttpResponse
import datetime

#Definición de la vista:
def saludo(request):
    texto = """
    <html>
    <body>
    <h1>Hola mundo</h1>
    </body>
    </html>
    """
    return HttpResponse(texto)

#Definición de una vista para contenido dinámico
def fecha(request):
    miFecha=datetime.datetime.now()

    texto2 = """<html>
    <body>
    <h2>Fecha y hora actual: </h2>%s
    </body>
    </html>
    """ % miFecha
    return HttpResponse(texto2)
```

Ahora debemos crear la URL que apunte a esta vista. Para ello en el archivo `urls.py`:

```
from django.contrib import admin
from django.urls import path
from miProyecto.views import saludo
from miProyecto.views import fecha

urlpatterns = [
    path('admin/', admin.site.urls),
    path('saludo/', saludo),
    path('fecha/', fecha),
]
```

Con lo que en la URL de `fecha` obtendremos en nuestro navegador:



Vamos a realizar otro ejemplo. Crearemos un script que calcule la edad que tendremos en un año determinado. En este ejemplo usaremos paso de parámetros por URL.

Creemos la vista:

```
from django.http import HttpResponse

def calcEdad(request, year):
    edadActual=18
    periodo=year-2022
    edadFutura=edadActual+periodo
    documento="<html><body><h2>En el año
%s tendrás %s años.</h2></body></html>"
    %(year, edadFutura)

    return HttpResponse(documento)
```

Ahora crearemos la URL. Necesitamos pasarle el parámetro `year`, eso lo hacemos con la sintaxis:

```
path('calcEdad/<year>', calcEdad),
```

Pero hay que tener en cuenta que todo parámetro que pasemos por URL se considera texto. Por lo que hay que pasarlo a número para poder operar con él.

```
path('calcEdad/<int:year>', calcEdad),
```

Por lo que nuestros scripts quedarían de la siguiente forma:

Archivo `views.py`:

```
from django.http import HttpResponse
import datetime

#Definición de la vista:
def saludo(request):
    texto = ""
    <html>
    <body>
    <h1>Hola mundo</h1>
    </body>
    </html>
    """
    return HttpResponse(texto)

#Definición de una vista para contenido
dinámico
def fecha(request):
    miFecha=datetime.datetime.now()
    texto2 = ""<html>
    <body>
    <h2>Fecha y hora actual: </h2>%s
    </body>
    </html>
    """ % miFecha
    return HttpResponse(texto2)
```

```
#Vista para calcular la edad que
tendremos en un año determinado
def calcEdad(request, year):
    edadActual=18
    periodo=year-2022
    edadFutura=edadActual+periodo
    documento="<html><body><h2>En el año
%s tendrás %s años.</h2></body></html>"
    %(year, edadFutura)

    return HttpResponse(documento)
```

Archivo `urls.py`:

```
from django.contrib import admin
from django.urls import path
from miProyecto.views import saludo,
fecha, calcEdad
urlpatterns = [
    path('admin/', admin.site.urls),
    path('saludo/', saludo),
    path('fecha/', fecha),
    path('calcEdad/<int:year>',
calcEdad),
]
```

Ahora ya podemos ver el resultado en nuestro navegador.

Importante poner la URL correctamente y pasarle el argumento del año:



Vamos a ver ahora cómo haríamos para pasar dos parámetros. En nuestro caso vamos a pasar ahora tanto `edadActual` como `year` por parámetros de URL.

Archivo `urls.py`:

```
from django.contrib import admin
from django.urls import path
from miProyecto.views import saludo
from miProyecto.views import fecha
from miProyecto.views import calcEdad

urlpatterns = [
    path('admin/', admin.site.urls),
    path('saludo/', saludo),
    path('fecha/', fecha),
    path('calcEdad/<int:edadActual>/<int:
year>', calcEdad),
]
```

Archivo `views.py`:

```
from django.http import HttpResponse
import datetime

#Definición de la vista:
def saludo(request):
    texto = """
<html>
<body>
<h1>Hola mundo</h1>
</body>
</html>
"""
    return HttpResponse(texto)

#Definición de una vista para contenido
dinámico
def fecha(request):
    miFecha=datetime.datetime.now()
    texto2 = """<html>
<body>
<h2>Fecha y hora actual: </h2>%s
</body>
</html>
""" % miFecha
    return HttpResponse(texto2)
```

```
#Vista para calcular la edad que  
tendremos en un año determinado  
def calcEdad(request, edadActual, year):  
    periodo=year-2022  
    edadFutura=edadActual+periodo  
    documento="<html><body><h2>En el año  
%s tendrás %s años.</h2></body></html>"  
    %(year, edadFutura)  
  
    return HttpResponseRedirect(documento)
```

Resultado en el navegador:

