



UANL

DOI-019
Vigencia 03-09/17

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN □ DIRECCIÓN DE TECNOLOGÍAS DE INFORMACIÓN

MATERIAL DE APOYO DE PYTHON NIVEL BÁSICO E INTERMEDIO



Índice

INTRODUCCIÓN A PYTHON	5
VENTAJAS DEL USO DE PYTHON.....	5
REQUERIMIENTOS E INSTALACIÓN DE PYTHON	6
TIPOS DE DATOS EN PYTHON	11
CADENAS	13
LISTAS	15
BOOLEANO	16
FECHA	17
ESTRUCTURAS DE CONTROL:	18
SENTENCIA IF	18
SENTENCIA FOR:	20
SENTENCIA WHILE	21
FUNCIONES	23
ARREGLOS.....	26
ARREGLOS UNIDIMENSIONALES	26
OPERACIONES CON ARREGLOS	26
LISTAS	28
TUPLAS	29
DICCIONARIOS	30
ACCESO A LISTAS, TUPLAS Y DICCIONARIOS.....	31
PROGRAMACIÓN ORIENTADA A OBJETOS	33
ELEMENTOS Y CARACTERÍSTICAS DE LA POO	33
CLASES	33
EJERCICIO PARA INICIAR CON EL TEMA (INSTRUCTOR Y ALUMNOS)	34
PROPIEDADES	36
MÉTODOS	37
OBJETOS	38
HERENCIA	39
ACCEDIENDO A LOS MÉTODOS Y PROPIEDADES DE UN OBJETO	41
CASOS DE USO DE LAS CLASES	42
EJERCICIO PARA REFORZAR	43



MÓDULOS Y PAQUETES	47
MÓDULOS DE PYTHON	48
IMPORTACIÓN DE MÓDULOS Y PAQUETES	50
CREACIÓN DE MÓDULOS	50
USAR MÓDULOS COMO PROGRAMAS	51
INSTALAR MÓDULOS EXTERNOS EN PYTHON	51
PAQUETES.....	54
ARCHIVOS.....	55
LECTURA	55
ESCRITURA.....	58
CIERRE DE ARCHIVOS:.....	58
BASES DE DATOS.....	59
HERRAMIENTA A UTILIZAR EN LA SESIÓN	59
CONCEPTOS DE BASE DE DATOS	59
CREAR UNA TABLA EN MYSQL	60
DISEÑO DE BASES DE DATOS.....	64
CREAR EL MODELO	65
CREAR LA TABLA	66
CÓMO AGREGAR CAMPOS A LA TABLA	66
CÓMO CREAR RELACIONES ENTRE TABLAS	67
CÓMO CREAR LAS TABLAS EN LA BASE DE DATOS A PARTIR DEL MODELO.....	68
COMANDOS BÁSICOS MYSQL	73
CONEXIÓN DE BASES DE DATOS CON PYTHON	78
INTRODUCCIÓN A BASES DE DATOS CON PYTHON	78
INSERTAR FILAS EN LA BASE DE DATOS.....	81
LEER BASE DE DATOS.....	83
ELIMINAR DATOS EN LA BASE DE DATOS CON PYTHON.....	86
INTRODUCCIÓN A HTML	87
ETIQUETAS.....	87
ESTRUCTURA	88
DISEÑO DE UNA PÁGINA WEB	89
CÓMO CREAR LISTAS EN PÁGINAS WEB	89
CÓMO USAR TABLAS EN PÁGINAS WEB.....	90
INSERTAR ENLACES (LINKS) EN PÁGINAS HTML	90
INSERTAR IMÁGENES EN UNA PÁGINA HTML.....	91
CÓMO PONER COLOR AL TEXTO EN HTML	91



PROGRAMACIÓN WEB CON DJANGO.....	93
EL SERVIDOR DE DESARROLLO	97
PUNTOS IMPORTANTES	98
EJERCICIO.....	99
ESCRIBA SU PRIMERA VISTA	100
CONFIGURACIÓN DE LA BASE DE DATOS	104
PARA BASES DE DATOS DISTINTAS A SQLITE.....	104
CREANDO MODELOS	106
ACTIVANDO LOS MODELOS.....	108
JUGANDO CON LA API.....	111
AGREGANDO NUEVAS VISTAS.....	114
USANDO TEMPLATES EN HTML.....	117
ACCEDER A BASE DE DATOS	119
FORMATEANDO NUESTRO RESULTADO	121
TAG EN HTML	122



Introducción a Python

Python es un lenguaje de programación poderoso y fácil de aprender. Cuenta con estructuras de datos eficientes y de alto nivel y un enfoque simple pero efectivo a la programación orientada a objetos. La elegante sintaxis de Python y su tipado dinámico, junto con su naturaleza interpretada, hacen de éste un lenguaje ideal para scripting y desarrollo rápido de aplicaciones en diversas áreas y sobre la mayoría de las plataformas.

Ventajas del uso de Python

El intérprete de Python y la extensa biblioteca estándar están a libre disposición en forma binaria y de código fuente para las principales plataformas desde el sitio web de Python, <https://www.python.org/>, y puede distribuirse libremente. El mismo sitio contiene también distribuciones y enlaces de muchos módulos libres de Python de terceros, programas y herramientas, y documentación adicional.

El intérprete de Python puede extenderse fácilmente con nuevas funcionalidades y tipos de datos implementados en C o C++ (u otros lenguajes accesibles desde C). Python también puede usarse como un lenguaje de extensiones para aplicaciones personalizables.

Python es fácil de usar, pero es un lenguaje de programación de verdad, ofreciendo mucha más estructura y soporte para programas grandes de lo que pueden ofrecer los scripts de Unix o archivos por lotes. Por otro lado, Python ofrece mucho más chequeo de error que C, y siendo un lenguaje de muy alto nivel, tiene tipos de datos de alto nivel incorporados como arreglos de tamaño flexible y diccionarios. Debido a sus tipos de datos más generales Python puede aplicarse a un dominio de problemas mayor que Awk o incluso Perl, y aun así muchas cosas siguen siendo al menos igual de fácil en Python que en esos lenguajes.

Python te permite separar tu programa en módulos que pueden reusarse en otros programas en Python. Viene con una gran colección de módulos estándar que puedes usar como base, o como ejemplos para empezar a aprender a programar en Python.

Python permite escribir programas compactos y legibles. Los programas en Python son típicamente más cortos que sus programas equivalentes en C, C++ o Java por varios motivos:

- los tipos de datos de alto nivel permiten expresar operaciones complejas en una sola instrucción
- la agrupación de instrucciones se hace por sangría en vez de llaves de apertura y cierre
- no es necesario declarar variables ni argumentos.

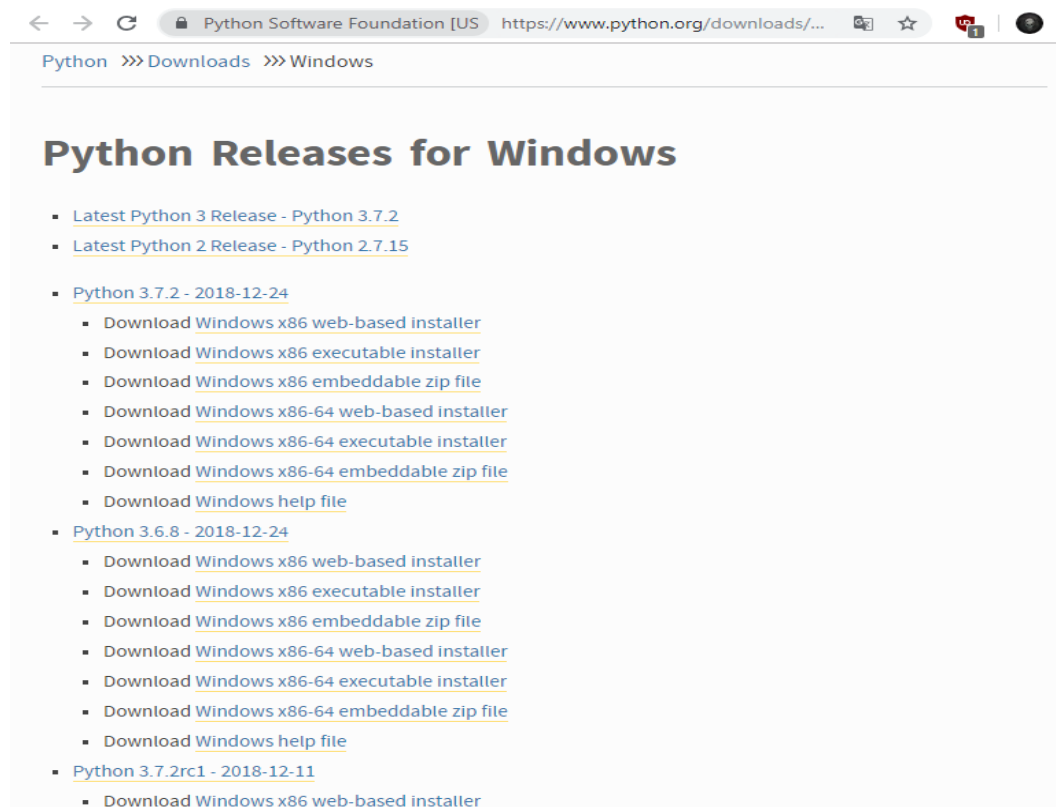
Python es extensible: si ya sabes programar en C es fácil agregar una nueva función o módulo al intérprete, ya sea para realizar operaciones críticas a velocidad máxima, o para enlazar programas Python con bibliotecas que tal vez sólo estén disponibles en forma binaria (por ejemplo, bibliotecas gráficas específicas de un fabricante).

Requerimientos e instalación de Python

Primero comprueba si tu computadora tiene una versión de Windows de 32-bit o 64-bit, en las propiedades del sistema operativo, en este caso Windows

Descargue el instalador de Python desde el sitio web:

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



Puedes descargar, de preferencia el instalador Windows x86-x64 executable installer, que es para las 2 versiones del sistema operativo.

Ejecuta el instalador, dando doble clic al archivo, ya terminada la descarga del sitio.

Sigue las instrucciones:

Al abrir el archivo, aparece la siguiente ventana:



Antes de dar clic en Install now, debemos marcar 2 casillas al final de esa ventana, que nos permitirán usar Python en la consola de Windows (CMD), como una variable de entorno.

Ya que están marcadas esas casillas, procede a dar clic en install now, y espera a que la paquetería se instale, para que des clic en el botón finalizar.

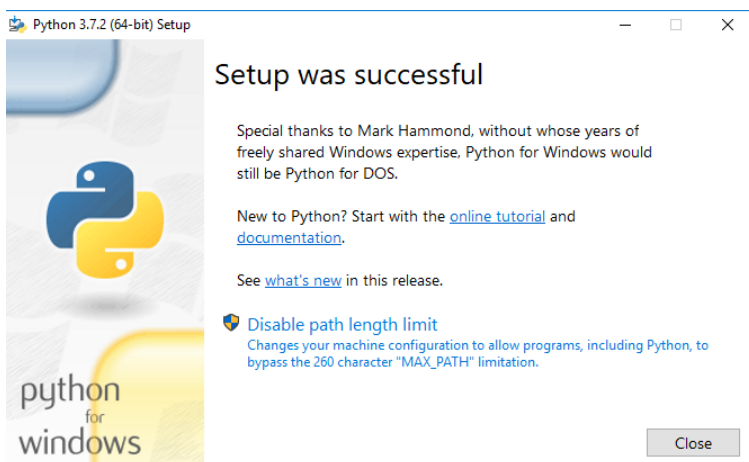
Ya instalado el paquete de Python, procederemos a comprobar que esta como variable de entorno, en la consola de comandos de Windows o CMD



UANL

DOI-019
Vigencia 03-09/17

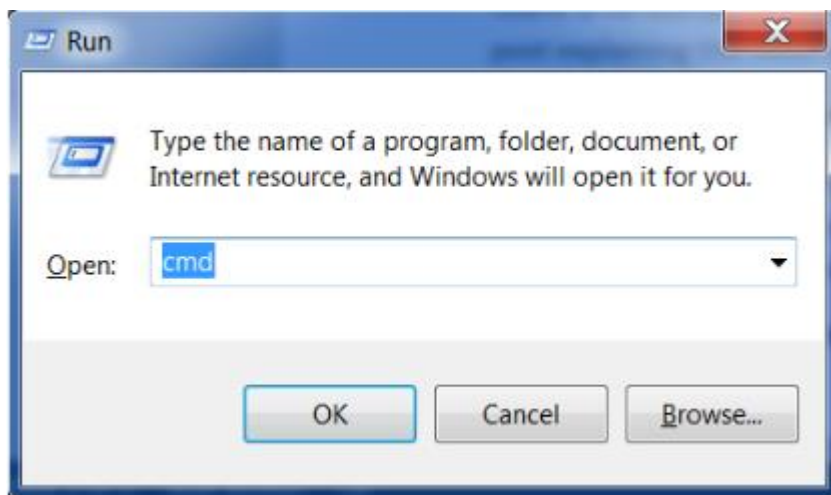
UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN ☐ DIRECCIÓN DE TECNOLOGÍAS DE INFORMACIÓN



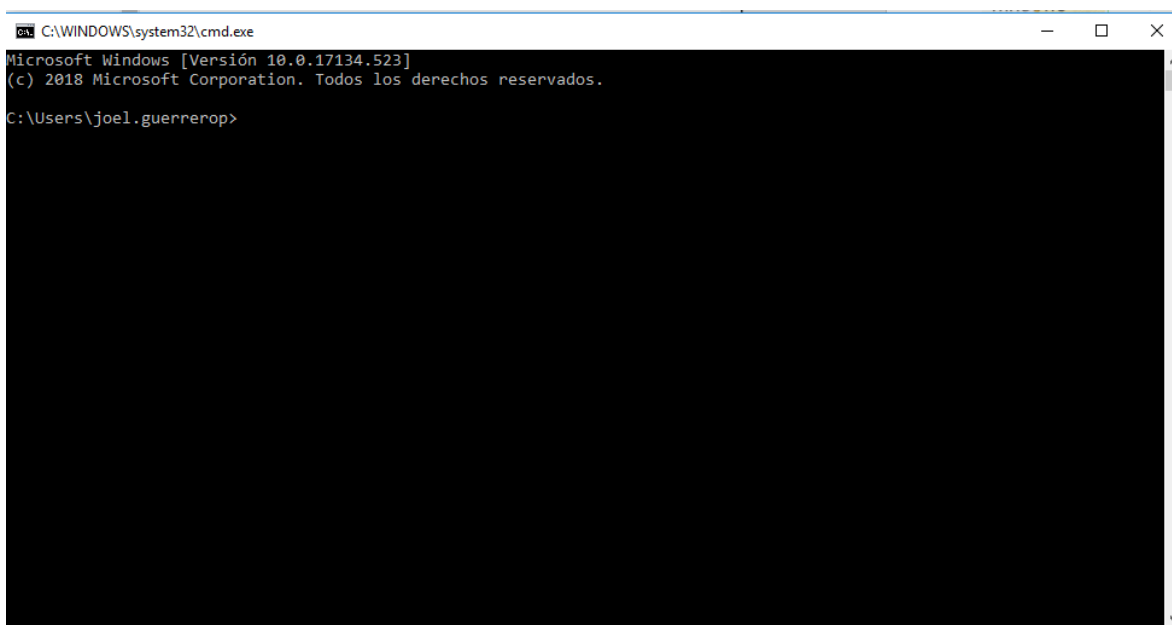
Para acceder al CMD, simplemente presiona los siguientes botones del teclado:

Botón Windows + letra R

Debe aparecer esta pequeña ventana, que se llama Ejecutar:

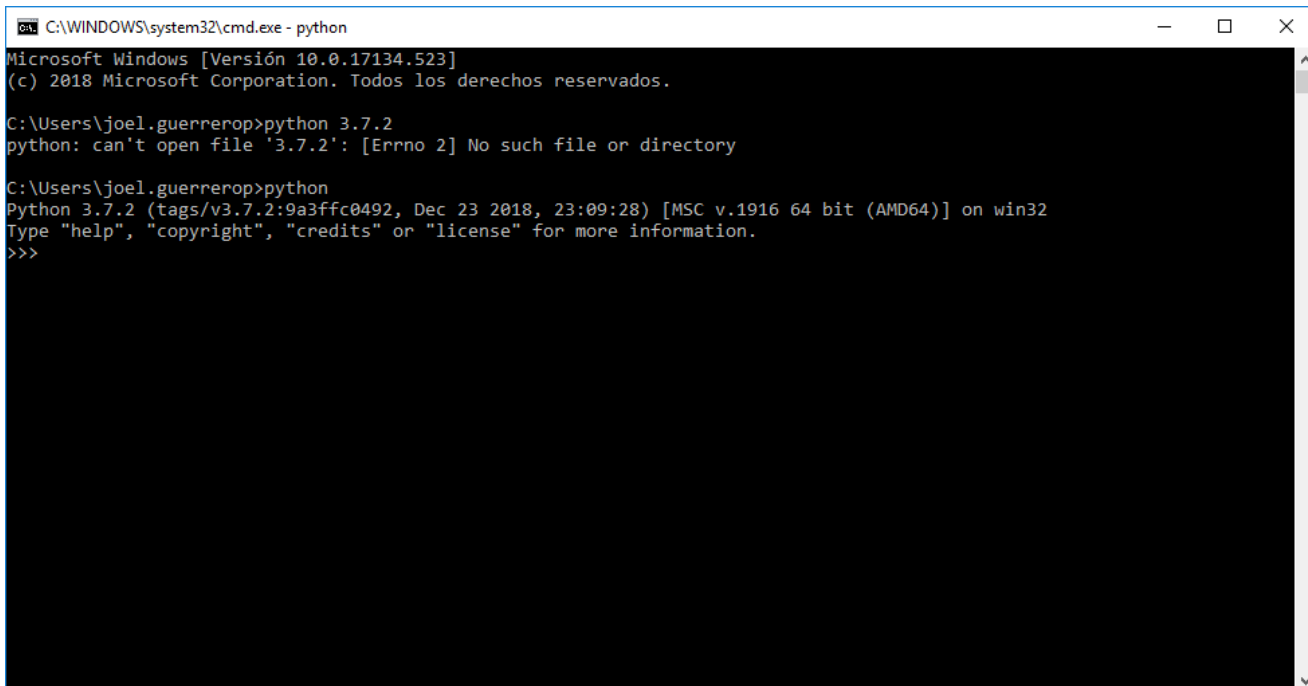


Teclea CMD, y presiona enter o da clic en aceptar, y aparecerá esta ventana:



En esta ventana, para comprobar que la instalación fue un éxito, teclea Python

Aparece lo siguiente en la ventana. Si se escribe Python junto con la versión, te marcara error, como en el primer renglón:



```
C:\WINDOWS\system32\cmd.exe - python
Microsoft Windows [Versión 10.0.17134.523]
(c) 2018 Microsoft Corporation. Todos los derechos reservados.

C:\Users\joel.guerrero>python 3.7.2
python: can't open file '3.7.2': [Errno 2] No such file or directory

C:\Users\joel.guerrero>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 23:09:28) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
```

Recuerda que solo debes teclear Python, para comprobar que esta agregado como variable de entorno en el equipo PC

Ya hecha la validación de que Python está instalado, ya podremos comenzar a trabajar con este lenguaje, y poder probar nuestros primeros programas.



Tipos de datos en Python

Primeramente, iniciaremos en cómo se codifica en Python.

Muchos de los ejemplos de este manual, incluso aquellos ingresados en el prompt interactivo, incluyen comentarios.

Los comentarios en Python comienzan con el carácter numeral, #, y se extienden hasta el final físico de la línea. Un comentario quizás aparezca al comienzo de la línea o seguidos de espacios blancos o código, pero no dentro de una cadena de caracteres.

Un carácter numeral dentro de una cadena de caracteres es sólo un carácter numeral. Ya que los comentarios son para aclarar código y no son interpretados por Python, pueden omitirse cuando se escriben los ejemplos.

Observa el siguiente ejemplo:

```
# este es el primer comentario
spam = 1                # y este es el segundo comentario
                        # ... y ahora un tercero!
text = "# Este no es un comentario".
```

En sublime Text, un editor de texto, recomendado para este lenguaje, se marcan los comentarios de Python en color gris.

Ahora, empecemos con el primer ejemplo, del primer tipo de dato que se ve en este lenguaje

Números.

Los números enteros son aquellos que no tienen decimales, tanto positivos como negativos (además del cero). En Python se pueden representar mediante el tipo int (de integer, entero) o el tipo long (largo). La única diferencia es que el tipo long permite almacenar números más grandes.

Aquellos con una parte fraccional (por ejemplo 5.0, 1.6) son de tipo float. Para aplicaciones normales puedes utilizar el tipo float sin miedo, como ha venido haciéndose desde hace años, aunque teniendo en cuenta que los números en coma flotante no son precisos (ni en este ni en otros lenguajes de programación).

Ejemplos sencillos de cómo usar este tipo de datos en Python

```
>>> 2 + 2
4
>>> 50 - 5*6
20
>>> (50 - 5*6) / 4
5.0
>>> 8 / 5 # la división siempre retorna un número de punto flotante
1.6
```



En este tipo de operación, en la división, se retorna en ocasiones a un punto flotante, o tipo float.

Para que el resultado sea entero, use el operador `//` o la llamada floor división

```
>>> 17 / 3 # la división clásica retorna un punto flotante
5.666666666666667
>>>
>>> 17 // 3 # la división entera descarta la parte fraccional
5
```

El signo igual (=) es usado para asignar valor a una variable. Observe el siguiente ejemplo:

```
>>> ancho = 20
>>> largo = 5 * 9
>>> ancho * largo
900
```

Errores comunes cuando se utiliza este tipo de datos:

Variable indefinida, que no está declarada o con un valor asignado

```
>>> n # tratamos de acceder a una variable no definida
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'n' is not defined
```

Ejemplo para practicar:

la última expresión impresa es asignada a la variable `_`. Esto significa que cuando estés usando Python como una calculadora de escritorio, es más fácil seguir calculando.

```
>>> impuesto = 12.5 / 100
>>> precio = 100.50
>>> precio * impuesto
12.5625
>>> precio + _
113.0625
>>> round(_, 2)
113.06
```

Esta variable debería ser tratada como de sólo lectura por el usuario. No le asignes explícitamente un valor; crearás una variable local independiente con el mismo nombre enmascarando la variable.



Cadenas

Python puede manipular cadenas de texto, las cuales pueden ser expresadas de distintas formas. Pueden estar encerradas en comillas simples ('...') o dobles ("...") con el mismo resultado. Observe el siguiente ejemplo:

```
>>> 'huevos y pan' # comillas simples
'huevos y pan'
>>> 'doesn\'t' # usa \' para escapar comillas simples...
"doesn't"
```

```
>>> "doesn't" # ...o de lo contrario usa comillas dobles
"doesn't"
>>> '"Si," le dijo.'
'"Si," le dijo.'
>>> "\'Si,\" le dijo."
'"Si," le dijo.'
>>> '"Isn\'t," she said.'
'"Isn\'t," she said.'
```

La salida de cadenas está encerrada en comillas y los caracteres especiales son escapados con barras invertidas. Aunque esto a veces luzca diferente de la entrada (las comillas que encierran pueden cambiar), las dos cadenas son equivalentes.

La cadena se encierra en comillas dobles si la cadena contiene una comilla simple y ninguna doble, de lo contrario es encerrada en comillas simples. La función print () produce una salida más legible, omitiendo las comillas que la encierran e imprimiendo caracteres especiales y escapados:

```
>>> '"Isn\'t," she said.'
'"Isn\'t," she said.'
>>> print('"Isn\'t," she said.')
"Isn't," she said.
>>> s = 'Primera línea.\nSegunda línea.' # \n significa nueva línea
>>> s # sin print(), \n es incluido en la salida
'Primera línea.\nSegunda línea.'
>>> print(s) # con print(), \n produce una nueva línea
Primera línea.
Segunda línea.
```

Las cadenas de texto pueden ser concatenadas (pegadas juntas) con el operador + y repetidas con *:

```
>>> # 3 veces 'un', seguido de 'ium'
>>> 3 * 'un' + 'ium'
'unununium'
```



Las cadenas de texto se pueden indexar (subíndices), el primer carácter de la cadena tiene el índice 0. No hay un tipo de dato para los caracteres; un carácter es simplemente una cadena de longitud uno:

```
>>> palabra = 'Python'
>>> palabra[0] # caracter en la posición 0
'P'
>>> palabra[5] # caracter en la posición 5
'n'
```

Los índices quizás sean números negativos, para empezar a contar desde la derecha:

```
>>> palabra[-1] # último caracter
'n'
>>> palabra[-2] # ante último caracter
'o'
>>> palabra[-6]
'P'
```

Además de los índices, las rebanadas también están soportadas. Mientras que los índices son usados para obtener caracteres individuales, las rebanadas te permiten obtener sub-cadenas:

```
>>> palabra[0:2] # caracteres desde la posición 0 (incluida) hasta la 2 (excluida)
'Py'
>>> palabra[2:5] # caracteres desde la posición 2 (incluida) hasta la 5 (excluida)
'tho'
```

Las cadenas de Python no pueden ser modificadas -- son inmutables. Por eso, asignar a una posición indexada de la cadena resulta en un error:

```
>>> palabra[0] = 'J'
...
TypeError: 'str' object does not support item assignment
>>> palabra[2:] = 'py'
...
TypeError: 'str' object does not support item assignment
```



Listas

Python tiene varios tipos de datos compuestos, usados para agrupar otros valores. El más versátil es la lista, la cual puede ser escrita como una lista de valores separados por coma (ítems) entre corchetes. Las listas pueden contener ítems de diferentes tipos, pero usualmente los ítems son del mismo tipo.

```
>>> cuadrados = [1, 4, 9, 16, 25]
>>> cuadrados
[1, 4, 9, 16, 25]
```

Las listas también soportan operaciones como concatenación:

```
>>> cuadrados + [36, 49, 64, 81, 100]
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

A diferencia de las cadenas de texto, que son inmutables, las listas son un tipo mutable, es posible cambiar un su contenido:

```
>>> cubos = [1, 8, 27, 65, 125] # hay algo mal aquí
>>> 4 ** 3 # el cubo de 4 es 64, no 65!
64
```

```
>>> cubos[3] = 64 # reemplazar el valor incorrecto
>>> cubos
[1, 8, 27, 64, 125]
```

La función predefinida `len()` también sirve para las listas:

```
>>> letras = ['a', 'b', 'c', 'd']
>>> len(letras)
4
```



Booleano

El tipo booleano, en términos generales, es una forma de representación de datos binarios, es decir, dos valores posibles únicamente: 0 (cero) y 1 (uno). Los dos valores posibles de este tipo de dato se prestan a diversas interpretaciones: Sí/No, Encendido/Apagado, Verdadero/Falso, entre otras; todo aquello que represente dos estados contrarios. En Python y los demás lenguajes de programación, se ha optado por las constantes True (1, verdadero) y False (0, falso).

En este ejemplo veremos cómo Python considera a todo valor distinto de 0 como True, y al cero como un dato False:

```
>>> bool(5)
True
>>> bool(0)
False
>>> bool(-3)
True
```

Si ejecutas código con If o elif, se evalúa la expresión, y si es verdadera, se ejecutará el código. Este apartado solo es un ejemplo de muestra, y este tema lo veremos más adelante en la clase.

```
if 5: # bool(5) -> True
    print("Este código es ejecutado.")

if 0: # bool(0) -> False
    print("Nunca se ejecuta.")
```




Fecha

Hay muchos tipos de objetos diferentes que podemos usar en Python. Las fechas son uno de esos objetos que son infames en casi todos los lenguajes de programación. Los tipos de fecha son muy difíciles de averiguar, pero Python tiene varias formas de manipular fechas y horas.

Primero, para que Python manipule el formato de fecha, debemos importar el módulo nativo `datetime`. Este módulo contiene todos los métodos que necesitamos para deshacernos del problema de formato. Podemos importarlo con una simple declaración `import`. Estamos usando la declaración `from` para poder hacer referencia a las funciones sin tener que usar la notación de puntos.

```
>>> from datetime import datetime
```

Se usará 2/4/18 y para comenzar, se representará como una cadena de caracteres (string) simple.

```
>>> strDate = '2/4/18'
```

Tenemos que convertir esta cadena en un objeto `datetime` para que Python pueda entender que la cadena es en realidad una fecha. Una forma de hacerlo es usando el método `strptime`. Este método en el objeto `datetime` nos permite pasar una fecha y hora y usar varios operadores de formato diferentes para manipular su función.

Cambiaremos de 2/4/18 a Feb 04, 2018. Para hacer eso, la cadena original la pondremos como primer argumento para el método `strptime` (). Este método convierte una cadena en un objeto de fecha que Python puede entender. El método `strptime` () le permite pasar una cadena simple como 2/4/18 y una cadena de formato simple para definir dónde se encuentra el día, el mes y el año.

```
>>> objDate = datetime.strptime(strDate, '%m/%d/%y')
>>> objDate
datetime.datetime(2018, 2, 4, 0, 0)
```

Ahora que Python sabe que esta cadena es una fecha real, podemos dejarla tal como está.



Estructuras de control:

Sentencia If

La estructura de control if ... permite que un programa ejecute unas instrucciones cuando se cumplan una condición.

La primera línea contiene la condición a evaluar y es una expresión lógica. Esta línea debe terminar siempre por dos puntos (:)

```
3 a = 7
4
5 if ( a > 5) : print "La variable es mayor a 5!"
6
7 print "fin"
```

Sintaxis alternativa con Else. La sentencia Else, significa, De lo contrario se cumple sin evaluar ninguna expresión condicional y ejecuta el bloque de sentencias seguidas. En pocas palabras, Else nos permite ejecutar la misma condición, y mostrarnos otro resultado, si la primera condición no se cumple.

```
03 a = 10
04
05 if ( a != 10) :
06     print "La variable es diferente de 10!"
07 else:
08     print "La variable es igual a 10!"
09
10 print "fin"
```

En este ejemplo, el if representado (a != 10), significa A diferente o igual a 10



Ejemplo para practicar:

```
>>> x = int(input("Ingresa un entero, por favor: "))
Ingresa un entero, por favor: 42
>>> if x < 0:
...     x = 0
...     print('Negativo cambiado a cero')
... elif x == 0:
...     print('Cero')
... elif x == 1:
...     print('Simple')
... else:
...     print('Más')
...
'Mas'
```

Observe la palabra reservada elif. Es una abreviación de 'else if', y es útil para evitar código en exceso. Una secuencia if ... elif ... elif ... sustituye las sentencias switch o case encontradas en otros lenguajes.



Sentencia For:

En ocasiones, tenemos que repetir varias veces una determinada tarea hasta conseguir nuestro objetivo. La sentencia For de Python itera sobre los ítems de cualquier secuencia (una lista o una cadena de texto), en el orden que aparecen en la secuencia.

En el caso del for, no es posible realizar un bucle infinito. Se puede utilizar con cualquier objeto con el que se pueda iterar (ir saltando de elemento en elemento), como verás en este ejemplo:

La función Len, determina el numero de letras o caracteres que tiene la palabra en esa lista.

```
>>> # Midiendo cadenas de texto
... palabras = ['gato', 'ventana', 'defenestrado']
>>> for p in palabras:
...     print(p, len(p))
...
gato 4
ventana 7
defenestrado 12
```

Veamos otro ejemplo, usando la función enumerate, que enumera el orden de la lista que esta en el bucle.

Primero se imprime el listado que esta en la variable coches:

```
>>> coches = ('Ferrari', 'Tesla', 'BMW', 'Audi')
>>> for coche in coches:
>>>     print(coche)

Ferrari
Tesla
BMW
Audi
```

Agregando la función enumerate. Recordemos que Python y otros lenguajes de programación empiezan a enumerar desde la posición 0:



```
>>> coches = ('Ferrari', 'Tesla', 'BMW', 'Audi')
>>> for i, coche in enumerate(coches):
>>>     print(str(i) + " - " + coche)

0 - Ferrari
1 - Tesla
2 - BMW
3 - Audi
```

Sentencia While

En ocasiones, tenemos que repetir varias veces una determinada tarea hasta conseguir nuestro objetivo. La condición también puede ser una cadena de texto o una lista, de hecho, cualquier secuencia; cualquier cosa con longitud distinta de cero es verdadero, las secuencias vacías son falsas.

Los operadores estándar de comparación se escriben igual que en C: < (menor qué), > (mayor qué), == (igual a), <= (menor o igual qué), >= (mayor o igual qué) y != (distinto a).

Con el bucle while, hay que tener la precaución de no realizar un “bucle infinito”, que consiste en un bucle que nunca termina por un error en la programación.

```
1 >>> vuelta=1
2 >>> while vuelta<10:
3 >>>     print("Vuelta "+str(vuelta))
4 >>>     vuelta=vuelta+1
5 Vuelta 1
6 Vuelta 2
7 Vuelta 3
8 Vuelta 4
9 Vuelta 5
10 Vuelta 6
11 Vuelta 7
12 Vuelta 8
13 Vuelta 9
```

Podemos usar este bucle con la sentencia else. Vea el siguiente ejemplo:



```
promedio, total, contar = 0.0, 0, 0
mensaje = "Introduzca la nota de un estudiante (-1 para salir): "

grado = int(raw_input(mensaje))
while grado != -1:
    total = total + grado
    contar += 1
    grado = int(raw_input(mensaje))
else:
    promedio = total / contar
    print "Promedio de notas del grado escolar: " + str(promedio)
```

Veamos otro ejemplo con la sentencia break, que nos ayuda a cortar el ciclo, cuidando de que no se convierta en un ciclo infinito, tal como se mencionó al principio del tema.

```
variable = 10

while variable > 0:
    print 'Actual valor de variable:', variable
    variable = variable -1
    if variable == 5:
        break
```



Funciones

Una función, puede tener cualquier tipo de algoritmo y cualquier cantidad de ellos y, utilizar cualquiera de las características que hemos visto. No obstante, una buena práctica, indica que la finalidad de una función, debe ser realizar una única acción, reutilizable y por lo tanto, tan genérica como sea posible.

La definición de funciones se realiza mediante la instrucción `def` más un nombre de función descriptivo -para el cuál, aplican las mismas reglas que para el nombre de las variables- seguido de paréntesis de apertura y cierre. Como toda estructura de control en Python, la definición de la función finaliza con dos puntos (`:`) y el algoritmo que la compone, irá indentado con 4 espacios:

```
def mi_funcion():  
    # aquí el algoritmo
```

Una función, no es ejecutada hasta tanto no sea invocada. Para invocar una función, simplemente se la llama por su nombre:

```
def funcion():  
    return "Hola Mundo"  
  
frase = funcion()  
  
print frase
```

Como característica importante, una función consta de parámetros, que es un valor que la función espera recibir cuando sea llamada (invocada), a fin de ejecutar acciones en base al mismo. Una función puede esperar uno o más parámetros (que irán separados por una coma) o ninguno.

```
def mi_funcion(nombre, apellido):  
    # algoritmo
```



Si quisiera acceder a una de las variables locales, fuera de la función, obtenemos un error:

```
def mi_funcion(nombre, apellido):  
    nombre_completo = nombre, apellido  
    print nombre_completo  
  
# Retornará el error: NameError: name 'nombre' is not defined  
print nombre
```

Importante: Al llamar una función, siempre se le deben pasar sus argumentos en el mismo orden que los está esperando. Pero hay una forma de evitar esto, usando keywords

“Keywords”: argumentos esperados, como pares de claves=valor:

```
def saludar(nombre, mensaje='Hola'):  
    print mensaje, nombre  
  
saludar(mensaje="Buen día", nombre="Juancho")
```




Formas de retorno en Python

En Python, si es posible llamar a una función dentro de otra, de forma fija y de la misma manera que se la llamaría, desde fuera de dicha función:

```
def funcion():  
    return "Hola Mundo"  
  
def saludar(nombre, mensaje='Hola'):  
    print mensaje, nombre  
    print mi_funcion()
```

Si es posible **realizar dicha llamada, de manera dinámica**, es decir, **desconociendo el nombre de la función** a la que se deseará llamar. A este tipo de acciones, se las denomina **llamadas de retorno**. Para este proceso, tenemos 2 funciones nativas de Python

Locals() y globals()

Ambas retornan un diccionario. Locals retorna elementos de ámbito local, y Globals, retorna lo mismo, pero a nivel global.

Ejemplo:

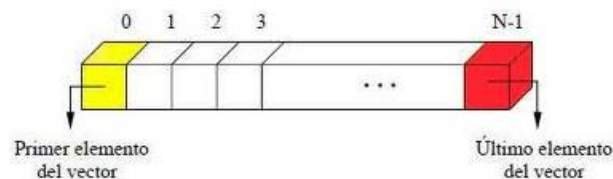
```
def funcion():  
    return "Hola Mundo"  
  
def llamada_de_retorno(func=""):  
    """Llamada de retorno a nivel global"""  
    return globals()[func]()  
  
print llamada_de_retorno("funcion")  
  
# Llamada de retorno a nivel local  
nombre_de_la_funcion = "funcion"  
print locals()[nombre_de_la_funcion]()
```

Arreglos

Un arreglo es un conjunto de datos o una estructura de datos homogéneos que se encuentran ubicados en forma consecutiva en la memoria RAM (sirve para almacenar datos en forma temporal).

Arreglos unidimensionales

Es un tipo de datos estructurado que está formado de una colección finita y ordenada de datos del mismo tipo. Es la estructura natural para modelar listas de elementos iguales. Están formados por un conjunto de elementos de un mismo tipo de datos que se almacenan bajo un mismo nombre, y se diferencian por la posición que tiene cada elemento dentro del arreglo de datos. Al declarar un arreglo, se debe inicializar sus elementos antes de utilizarlos. Para declarar un arreglo tiene que indicar su tipo, un nombre único y la cantidad de elementos que va a contener.



Operaciones con arreglos

Las operaciones en arreglos pueden clasificarse de la siguiente forma:

- Lectura: este proceso consiste en leer un dato de un arreglo y asignar un valor a cada uno de sus componentes
- Escritura: Consiste en asignarle un valor a cada elemento del arreglo.
- Asignación: No es posible asignar directamente un valor a todo el arreglo



- Actualización: Dentro de esta operación se encuentran las operaciones de eliminar, insertar y modificar datos. Para realizar este tipo de operaciones se debe tomar en cuenta si el arreglo está o no ordenado.
- Ordenación.
- Búsqueda.
- Insertar.
- Borrar.
- Modificar.



Listas

Una lista es una estructura de datos y un tipo de dato en python con características especiales. Lo especial de las listas en Python es que nos permiten almacenar cualquier tipo de valor como enteros, cadenas y hasta otras funciones; por ejemplo:

```
mylist = [1, 2, 3, 4, 5]
```

Las listas son mutables porque los elementos se pueden cambiar o reordenar.

Si tenemos una lista como la siguiente:

```
mylist = ['one', 'two', 'three', 'four', 'five']
```

Podemos cambiar el tercer elemento de la siguiente forma:

```
mylist[2] = "New item"
```

Ahora, si imprimes la lista, deberías obtener una nueva con el elemento modificado:

```
['one', 'two', 'New item', 'four', 'five']
```

Si el índice es negativo, cuenta desde el último elemento.

```
mylist = ['one', 'two', 'three', 'four', 'five']  
elem = mylist[-1]  
print(elem)
```

El resultado es 5



Tuplas

Las tuplas son secuencias, igual que las cadenas, y se puede utilizar la misma notación de índices que en las cadenas para obtener cada una de sus componentes.

Ejemplo:

- El primer elemento de (25, "Mayo", 1810) es 25.
- El segundo elemento de (25, "Mayo", 1810) es "Mayo".
- El tercer elemento de (25, "Mayo", 1810) es 1810.

```
>>> t=(25, "Mayo", 1810)
>>> t[0]
25
>>> t[1]
'Mayo'
>>> t[2]
1810
```

Las tuplas son inmutables, o mejor dicho, no pueden ser modificadas:

```
>>> t[2] = 2008
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Ejemplo donde se puso un valor diferente al registrado, y lo marco como error



Diccionarios

[u]n diccionario es una obra de consulta de palabras y/o términos que se encuentran generalmente ordenados alfabéticamente. Al igual que los diccionarios a los que usamos habitualmente en la vida diaria, los diccionarios de Python son una lista de consulta de términos de los cuales se proporcionan valores asociados.

En Python, un diccionario es una colección no-ordenada de valores que son accedidos a través de una clave. Es decir, en lugar de acceder a la información mediante el índice numérico, como es el caso de las listas y tuplas, es posible acceder a los valores a través de sus claves, que pueden ser de diversos tipos.

Las claves son únicas dentro de un diccionario, es decir que no puede haber un diccionario que tenga dos veces la misma clave, si se asigna un valor a una clave ya existente, se reemplaza el valor anterior.

No hay una forma directa de acceder a una clave a través de su valor, y nada impide que un mismo valor se encuentre asignado a distintas claves

La información almacenada en los diccionarios, no tiene un orden particular. Ni por clave ni por valor, ni tampoco por el orden en que han sido agregados al diccionario.

Cualquier variable de tipo inmutable, puede ser clave de un diccionario: cadenas, enteros, tuplas (con valores inmutables en sus miembros), etc. No hay restricciones para los valores que el diccionario puede contener, cualquier tipo puede ser el valor: listas, cadenas, tuplas, otros diccionarios, objetos, etc.

Para definirlo junto con los miembros que va a contener, se encierra el listado de valores entre llaves, las parejas de clave y valor se separan con comas, y la clave y el valor se separan con (:)

```
punto = {'x': 2, 'y': 1, 'z': 4}
```

Nota:

El algoritmo que usa Python internamente para buscar un elemento en un diccionario es muy distinto que el que utiliza para buscar en listas.

Para buscar en las listas, se utiliza un algoritmo de comparación que tarda cada vez más a medida que la lista se hace más larga. En cambio, para buscar en diccionarios se utiliza un algoritmo llamado hash, que se basa en realizar un cálculo numérico sobre la clave del elemento, y tiene una propiedad muy interesante: sin importar cuántos elementos tenga el diccionario, el tiempo de búsqueda es siempre aproximadamente igual.



Este algoritmo de hash es también la razón por la cual las claves de los diccionarios deben ser inmutables, ya que la operación hecha sobre las claves debe dar siempre el mismo resultado, y si se utilizara una variable mutable esto no sería posible.

Acceso a listas, tuplas y diccionarios

Puedes acceder a los distintos elementos de una lista o tupla indicando el índice (comenzando desde el 0) entre corchetes.

```
>>> a = ["Hola", "mundo", "!"]
>>> a[0]
'Hola'
>>> a[1]
'mundo'
>>> a[2]
'!'
```

Si el índice está fuera del rango, obtendrás una excepción.

```
>>> a[3]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

Ya que `a` contiene tres elementos: el 0, el 1, y el 2.

Remover elementos

El método `remove` de una lista eliminará el primer elemento que coincida con el valor indicado como argumento.

```
>>> a = ["Perro", "Gato", "Caballo"]
>>> a.remove("Gato")
>>> a
['Perro', 'Caballo']
```

Si el elemento no se encuentra en la lista, se lanzará `ValueError`.

```
>>> a.remove("Pez")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: list.remove(x): x not in list
```



Acceso a diccionarios

Para acceder a alguno de los valores se indica su clave correspondiente entre corchetes.

```
>>> d = {"Python": 1991, "C": 1972, "Java": 1996}

>>> d["Python"]
1991
```

Y bajo la misma sintaxis le asignamos un nuevo valor.

```
>>> d["Python"] = 2001
>>> d["Python"]
2001
```

También de esta forma podemos agregar nuevos elementos.

```
>>> d["C++"] = 1983
>>> d
{'C++': 1983, 'Java': 1996, 'Python': 2001, 'C': 1972}
```

Por cuanto las claves actúan como identificadores, no es posible que haya dos iguales (si esto fuese posible Python no sabría cuál de sus valores asociados debería retornar). No obstante, puede haber dos claves (o más) con el mismo valor.

```
>>> d["JavaScript"] = 1995
>>> d["PHP"] = 1995
```

Cuando intentamos acceder a una clave inexistente, se lanza la excepción `KeyError`.

```
>>> d["Basic"]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'Basic'
```




Programación Orientada a Objetos

La Programación Orientada a Objetos (POO u OOP por sus siglas en inglés), es un paradigma de programación.

Paradigma: teoría cuyo núcleo central [...] suministra la base y modelo para resolver problemas [...] (Definición de la Real Academia Española, vigésimo tercera edición)

Cómo tal, nos enseña un método -probado y estudiado- el cual se basa en las interacciones de objetos (todo lo descrito en el título anterior, Pensar en objetos) para resolver las necesidades de un sistema informático.

Elementos y Características de la POO

Los elementos de la POO, pueden entenderse como los materiales que necesitamos para diseñar y programar un sistema, mientras que las características, podrían asumirse como las herramientas de las cuáles disponemos para construir el sistema con esos materiales.

Clases

Las clases proveen una forma de empaquetar datos y funcionalidad juntos. Al crear una nueva clase, se crea un nuevo tipo de objeto, permitiendo crear nuevas instancias de ese tipo. Cada instancia de clase puede tener atributos adjuntos para mantener su estado. Las instancias de clase también pueden tener métodos (definidos por su clase) para modificar su estado.

En pocas palabras, las clases son una forma para crear sus propios objetos. Los objetos son una encapsulación de variables y funciones en una sola entidad. Los objetos obtienen sus variables y funciones de las clases. Es una excelente manera de conceptualizar, organizar y construir una jerarquía para cualquier organización o proceso.



Ejercicio para iniciar con el tema (instructor y alumnos)

```
1 class perro():
2
3     def __init__(self, nombre, size, raza):
4         self.nombre=nombre
5         self.size=size
6         self.raza=raza
7
8     def ladra(self):
9         s=" "
10        for l in self.nombre:
11            s+="wof"
12        print (s)
13
14
15 chucho=perro("chucho","grande","husky")
16 chucho.ladra()
17
18 #wofwofwofwofwofwof
19
20 cloe=perro("cloe","mini","chihuahua")
21 cloe.ladra()
22
23 #wofwofwofwof|
```

Ejemplo de una clase. El resultado de esta clase, si se compila en CMD, son los comentarios que están en el código



En Python, una clase se define con la instrucción `class` seguida de un nombre genérico para el objeto.

En este ejemplo, Imagina que tienes un juguete. Se crea una especie (“clase”).

```
class Objeto:
```

```
    pass
```

```
class Antena:
```

```
    pass
```

```
class Pelo:
```

```
    pass
```

```
class Ojo:
```

```
    pass
```

Recordatorio: los nombres de instancias se escriben en letras minúsculas. Por el contrario, las clases se nombran sin guiones bajos y poniendo en mayúscula la primera letra de cada palabra.



Propiedades

Las propiedades, como hemos visto antes, son las características intrínsecas del objeto. Éstas, se representan a modo de variables, solo que técnicamente, pasan a denominarse propiedades

En pocas palabras, atributos para esa clase: peso, forma, color, tamaño, aspecto, antena etc.

```
class Antena():
    color = ""
    longitud = ""

class Pelo():
    color = ""
    textura = ""

class Ojo():
    forma = ""
    color = ""
    tamaño = ""

class Objeto():
    color = ""
    tamaño = ""
    aspecto = ""
    antenas = Antena() # propiedad compuesta por el objeto objeto Antena
    ojos = Ojo()        # propiedad compuesta por el objeto objeto Ojo
    pelos = Pelo()      # propiedad compuesta por el objeto objeto Pelo
```



Métodos

Los métodos son funciones (como las que vimos en el capítulo anterior), solo que técnicamente se denominan métodos, y representan acciones propias que puede realizar el objeto (y no otro)

```
class Objeto():  
    color = "verde"  
    tamaño = "grande"  
    aspecto = "feo"  
    antenas = Antena()  
    ojos = Ojo()  
    pelos = Pelo()  
  
    def flotar(self):  
        pass
```



Objetos

Las clases por sí mismas, no son más que modelos que nos servirán para crear objetos en concreto. Podemos decir que una clase, es el razonamiento abstracto de un objeto, mientras que el objeto, es su materialización. A la acción de crear objetos, se la denomina instanciar una clase y dicha instancia, consiste en asignar la clase, como valor a una variable:

```
class Objeto():  
    color = "verde"  
    tamaño = "grande"  
    aspecto = "feo"  
    antenas = Antena()  
    ojos = Ojo()  
    pelos = Pelo()  
  
    def flotar(self):  
        print 12  
  
et = Objeto()  
print et.color  
print et.tamaño  
print et.aspecto  
et.color = "rosa"  
print et.color
```



Herencia

Algunos objetos comparten las mismas propiedades y métodos que otro objeto, y además agregan nuevas propiedades y métodos. A esto se lo denomina herencia: una clase que hereda de otra.

Vale aclarar, que, en Python, cuando una clase no hereda de ninguna otra, debe hacerse heredar de object, que es la clase principal de Python, que define un objeto.

```
class Antena(object):
```

```
    color = ""
```

```
    longitud = ""
```

```
class Pelo(object):
```

```
    color = ""
```

```
    textura = ""
```

```
class Ojo(object):
```

```
    forma = ""
```

```
    color = ""
```

```
    tamaño = ""
```

```
class Objeto(object):
```

```
    color = ""
```

```
    tamaño = ""
```

```
    aspecto = ""
```

```
    antenas = Antena()
```

```
    ojos = Ojo()
```

```
    pelos = Pelo()
```



```
def flotar(self):  
    pass  
  
class Dedo(object):  
    longitud = ""  
    forma = ""  
    color = ""  
  
class Pie(object):  
    forma = ""  
    color = ""  
    dedos = Dedo()  
  
# NuevoObjeto si hereda de otra clase: Objeto  
class NuevoObjeto(Objeto):  
    pie = Pie()  
  
    def saltar(self):  
        pass
```




Accediendo a los métodos y propiedades de un objeto

Una vez creado un objeto, es decir, una vez hecha la instancia de clase, es posible acceder a su métodos y propiedades. Para ello, Python utiliza una sintaxis muy simple: el nombre del objeto, seguido de punto y la propiedad o método al cuál se desea acceder:

```
objeto = MiClase()
print objeto.propiedad
objeto.otra_propiedad = "Nuevo valor"
variable = objeto.metodo()
print variable
print objeto.otro_metodo()
```



Casos de uso de las clases

```
>>> class Calculadora (object):  
    def __init__(self, operando1, operando2):  
        self.op1 = operando1  
        self.op2 = operando2  
    def sumar (self):  
        return self.op1 + self.op2  
    def restar (self):  
        return self.op1 - self.op2  
    def multiplicar (self):  
        return self.op1 * self.op2  
    def dividir (self):  
        return self.op1 / self.op2
```

Constructor
Recibe por parámetros a operando1 y operando2, con los que se inicializa a los atributos op1 y op2.
ATENCIÓN al 'self.'

```
>>> o = Calculadora (10, 2)  
>>> o.sumar ()  
12  
>>> o.restar ()  
8  
>>> o.multiplicar ()  
20  
>>> o.dividir ()  
5  
>>>
```

Creación de objeto con parametros 10 y 2

Llamadas a métodos de la clase.
Los valores con los que se trabaja son los atributos del objeto 'o'

```
>>> class Saludo (object):  
    nombre = 'Jose Luis'  
    apellidos = 'Montero Fuentes'  
    def saludar (self):  
        print 'Hola. Soy %s %s' % (self.nombre, self.apellidos)
```

Creación de objeto (instanciación)

```
>>> objeto = Saludo ()  
>>> objeto.saludar ()  
Hola. Soy Jose Luis Montero Fuentes  
>>>
```

Ejecución de método



Ejercicio para reforzar

Creamos un archivo .py con el nombre clase1.py

```
hola = "¡Hola mundo!"  
  
def imprimir():  
    print hola
```

Creamos otro archivo, con nombre mimodulo.py, con el siguiente código, para llamar a la clase anterior:

```
import mimodulo  
  
mimodulo.imprimir()  
print mimodulo.hola
```

La ejecución mostrará dos veces el mensaje ¡Hola mundo!

Sin entrar en detalles sobre la programación orientada a objetos, es posible ver a una clase como un módulo. Crea un archivo .py que se llame miclase.py

```
class miclase(object):  
  
    def __init__(self):  
        self.hola = "¡Hola mundo!"  
  
    def imprimir(self):  
        print self.hola
```

La diferencia al utilizar módulos o clases radica en que un módulo es instanciado una única vez (al importarlo), por lo cual tendremos en todo momento un único valor para la variable hola, mientras que una clase puede ser instanciada múltiples veces, donde cada instancia tendrá su propia copia de la variable hola (de allí el uso de self para definir variables propias de cada instancia). Lógicamente la clase se importa una única vez, pero es posible tener múltiples "instancias" (objetos) de la misma, y cada una no interfiere con las demás.

Esta instanciación (crear un nuevo objeto o instancia de la clase importada) se produce al llamar a la clase como si fuese una función (lo que implica que se ejecute la función __init__ de la misma):



```
1 def es_cadena_no_vacia(str):
2     if str == "":
3         return False
4     else:
5         return True
6 def es_numero(value):
7     try:
8         float(value)
9         return True
10    except ValueError:
11        return False
12
13 class Hotel(object):
14     """ Hotel: sus atributos son: nombre, ubicacion, puntaje y precio."""
15
16     def __init__(self, nombre = '', ubicacion = '',
17                 puntaje = 0, precio = float("inf")):
18         """ nombre y ubicacion deben ser cadenas no vacias,
19             puntaje y precio son números.
20             Si el precio es 0 se reemplaza por infinito. """
21
22         if es_cadena_no_vacia (nombre):
23             self.nombre = nombre
24         else:
25             raise TypeError ("El nombre debe ser una cadena no vacia")
26
27         if es_cadena_no_vacia (ubicacion):
28             self.ubicacion = ubicacion
29         else:
30             raise TypeError ("La ubicación debe ser una cadena no vacia")
31
32         if es_numero(puntaje):
33             self.puntaje = puntaje
34         else:
35             raise TypeError ("El puntaje debe ser un número")
36
37         if es_numero(precio):
38             if precio != 0:
39                 self.precio = precio
40             else:
41                 self.precio = float("inf")
42         else:
43             raise TypeError("El precio debe ser un número")
44
45     def __str__(self):
46         """ Muestra el hotel según lo requerido. """
47         return self.nombre + " de " + self.ubicacion + \
48             " - Puntaje: " + str(self.puntaje) + " - Precio: " + \
49             str(self.precio) + " pesos."
50 h= Hotel("Hotel City", "Mercedes", 3.25, 78)
51 print (h)
```



Ejercicio para reforzar

En base al código anterior, quite las 2 últimas líneas de código, y agregue las siguientes líneas que se muestran a continuación.

¿Qué le hace falta para que la lista de hoteles, se ordene? Se agregó el print del método ratio, para ayudar al ordenamiento de la lista de hoteles.



```
def ratio(self):
    return((self.puntaje*2)*10.)/(self.precio)

def __cmp__(self, otro):
    diferencia = self.ratio() - otro.ratio()
    if diferencia < 0:
        return -1
    elif diferencia > 0:
        return 1
    else:
        return 0

h1= Hotel('Hotel 1 *normal', 'mexico', 1, 10)
h2= Hotel("Hotel 2 *normal", "Mexico", 2, 40)
h3= Hotel("Hotel 3 *carisimo", "Mexico", 3, 130)
h4= Hotel("Hotel 4 *vale la pena", "mexico", 4, 130)
lista = [ h1, h2, h3, h4 ]

for hotel in lista:
    print (hotel)

print (h1.ratio())
print (h2.ratio())
print (h3.ratio())
print (h4.ratio())
```

Resultado sin ordenación:

```
Hotel 1 *normal de mexico - Puntaje: 1 - Precio: 10 pesos.
Hotel 2 *normal de Mexico - Puntaje: 2 - Precio: 40 pesos.
Hotel 3 *carisimo de Mexico - Puntaje: 3 - Precio: 130 pesos.
Hotel 4 *vale la pena de mexico - Puntaje: 4 - Precio: 130 pesos.
1.0
1.0
0.6923076923076923
1.2307692307692308
```



Módulos y paquetes

Si sales del intérprete de Python y entras de nuevo, las definiciones que hiciste (funciones y variables) se pierden. Por lo tanto, si quieres escribir un programa más o menos largo, es mejor que uses un editor de texto para preparar la entrada para el intérprete y ejecutarlo con ese archivo como entrada. Esto es conocido como crear un guión, o script. Si el programa se vuelve más largo, conviene separarlo en distintos archivos para un mantenimiento más fácil. O puedes usar una función útil que escribiste desde distintos programas sin copiar su definición a cada programa.

Para soportar esto, Python tiene una manera de poner definiciones en un archivo y usarlos en un script o en una instancia interactiva del intérprete. Tal archivo es llamado módulo; las definiciones de un módulo pueden ser importadas a otros módulos o al módulo principal (la colección de variables a las que tenemos acceso en un script ejecutado en el nivel superior y en el modo calculadora).

Un módulo es un archivo conteniendo definiciones y declaraciones de Python. El nombre del archivo es el nombre del módulo con el sufijo `.py` agregado. Dentro de un módulo, el nombre del mismo (como una cadena) está disponible en el valor de la variable global `__name__`.

Las ventajas de usar módulos son:

- Las funciones y variables deben ser definidas sólo una vez, y luego pueden ser utilizadas en muchos programas sin necesidad de reescribir el código;
- Permiten que un programa pueda ser organizado en varias secciones lógicas, puestas cada una en un archivo separado;
- Hacen más fácil compartir componentes con otros programadores.

Ejemplo:

```
1 from math import exp, cos
2 from math import pi, e
3 print (cos(pi / 3))
4 |
```

Resultado: 0.5



Módulos de Python

Éstos son algunos de los módulos estándares de Python, que pueden ser usado desde cualquier programa.

El módulo math contiene funciones y constantes matemáticas:

```
1  from math import floor, radians
2  floor(-5.9)
3  print(floor(-5.9))
4  radians(180)
5  print(radians(180))
```

El módulo random de la librería estándar de Python incluye un conjunto de funciones que permiten obtener de distintos modos números aleatorios o, para ser rigurosos, pseudoaleatorios.

```
1  import random
2
3  r=random.choice(['cara', 'sello'])
4  print(r)
5  s=random.choice(['cara', 'sello'])
6  print(s)
7  t=random.choice(['cara', 'sello'])
8  print(t)
9
10 u=random.randrange(10)
11 print(u)
12
13 |
14 w=[1,2,3,4]
15 random.shuffle(w)
16 print(w)
```

Ejemplo de cómo importar el módulo random, instalado en la paquetería de Python.



El módulo `datetime` provee tipos de datos para manipular fechas y horas:

```
1 from datetime import date
2 hoy = date(2011, 5, 31)
3 fin_del_mundo = date(2012, 12, 21)
4 print(fin_del_mundo - hoy)
```

El módulo `fractions` provee un tipo de datos para representar números racionales:

```
1 from fractions import Fraction
2 a = Fraction(5, 12)
3 b = Fraction(9, 7)
4 print(a + b)
5 |
```

El módulo `turtle` permite manejar una tortuga (¡haga la prueba!):

```
1 from turtle import Turtle
2 t = Turtle()
3 t.forward(10)
4 t.left(45)
5 t.forward(20)
6 t.left(45)
7 t.forward(30)
8 for i in range(10):
9     t.right(30)
10    t.forward(10 * i)
11
12 print(t)
13 |
```



Importación de módulos y paquetes

La sentencia `import` importa objetos desde un módulo para poder ser usados en el programa actual. Una manera de usar `import` es importar sólo los nombres específicos que uno desea utilizar en el programa:

```
1 import math
2 print (math.sin(10))
3 print (math.cos(20))
```

La otra manera de usar `import` es importando el módulo completo, y accediendo a sus objetos mediante un punto

```
1 from math import sin, cos
2 print (sin(10))
3 print (cos(20))
```

En este caso, las funciones `sin` y `cos` no fueron creadas por nosotros, sino importadas del módulo de matemáticas, donde están definidas.

En las dos formas de importar un módulo, las 2 son equivalentes. Hay que optar por la que hace que el programa sea más fácil de entender.

Creación de módulos

Un módulo sencillo es simplemente un archivo con código en Python. El nombre del archivo indica cuál es el nombre del módulo.

Ejemplo:

Hagamos un módulo, primero, crear un archivo en el editor de texto llamado `pares.py`

```
1 def es_par(n):
2     return n % 2 == 0
3
4 def es_impar(n):
5     return not es_par(n)
6
7 def pares_hasta(n):
8     return range(0, n, 2)
```

Después, crear otro archivo llamado `mostrar_pares.py` en el editor de texto, que importe el módulo creado anteriormente.

```
1 from pares import pares_hasta
2
3 n = int(input('Ingrese un entero: '))
4 print ('Los numeros pares hasta', n, 'son:')
5 for i in pares_hasta(n):
6     print (i)
```

Usar módulos como programas

Un archivo con extensión .py puede ser un módulo o un programa. Si es un módulo, contiene definiciones que pueden ser importadas desde un programa o desde otro módulo. Si es un programa, contiene código para ser ejecutado.

A veces, un programa también contiene definiciones (por ejemplo, funciones y variables) que también pueden ser útiles desde otro programa. Sin embargo, no pueden ser importadas, ya que al usar la sentencia import el programa completo sería ejecutado. Lo que ocurriría en este caso es que, al ejecutar el segundo programa, también se ejecutaría el primero.

Existe un truco para evitar este problema: siempre que hay código siendo ejecutado, existe una variable llamada `__name__`. Cuando se trata de un programa, el valor de esta variable es `'__main__'`, mientras que, en un módulo, es el nombre del módulo. Por lo tanto, se puede usar el valor de esta variable para marcar la parte del programa que debe ser ejecutada al ejecutar el archivo, pero no al importarlo.

Instalar módulos externos en Python

Al igual que con cualquier lenguaje de programación serio, Python admite bibliotecas y marcos de terceros que puede instalar para evitar tener que reinventar la rueda con cada nuevo proyecto. Puede encontrarlos en un repositorio central llamado "PyPI" (Python Package Index).

Pero descargar, instalar y administrar estos paquetes a mano puede ser frustrante y llevar mucho tiempo, por lo que muchos desarrolladores de Python confían en una herramienta especial llamada PIP para que Python haga todo mucho más fácil y rápido.

PIP es un acrónimo que significa "Paquetes de instalación PIP" o "Programa de instalación preferida". Es una utilidad de línea de comandos que le permite instalar, reinstalar o desinstalar paquetes PyPI con un comando simple y directo: "pip". Si está utilizando Python 2.7.9 (o superior) o Python 3.4 (o superior), entonces PIP viene instalado con Python por defecto.

Comprobar que pip está instalado en nuestra máquina. Abrir CMD para ejecutar estas instrucciones:

```
C:\Users\joel.guerrero>pip list
Package      Version
-----
pip          18.1
setuptools  40.6.2
You are using pip version 18.1, however version 19.0.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

C:\Users\joel.guerrero>-m pip install --upgrade pip
"-m" no se reconoce como un comando interno o externo,
programa o archivo por lotes ejecutable.

C:\Users\joel.guerrero>python -m pip install --upgrade pip
Collecting pip
  Downloading https://files.pythonhosted.org/packages/46/dc/7fd5df840efb3e56c8b4f768793a/pip-19.0.1-py2.py3-none-any.whl (1.4MB)
    100% |#####| 1.4MB 6.2MB/s
Installing collected packages: pip
  Found existing installation: pip 18.1
  Uninstalling pip-18.1:
    Successfully uninstalled pip-18.1
  Successfully installed pip-19.0.1
```

Una vez que PIP esté listo, puede comenzar a instalar paquetes de PyPI:



`pip install nombre-paquete`

Para instalar una versión específica de un paquete en lugar de la última versión:

`pip install nombre-paquete == 1.0.0`

Para buscar PyPI para un paquete particular:

`pip search "query"`

Para ver detalles sobre un paquete instalado:

`pip show nombre-paquete`

Para enumerar todos los paquetes instalados:

`pip list`

Para enumerar todos los paquetes desactualizados:

`pip list --outdated`

Para actualizar un paquete desactualizado:

`pip install nombre-paquete --upgrade`

Tenga en cuenta que las versiones anteriores de un paquete se eliminan automáticamente por PIP cuando se actualiza a una versión más nueva de ese paquete.

Para reinstalar completamente un paquete:

`pip install nombre-paquete -upgrade --force-reinstall`

Para deshacerse completamente de un paquete:

`pip uninstall nombre-paquete`

Si solo teclea pip en la consola de comandos, aparecen sus opciones generales



```
C:\Users\joel.guerrero>pip

Usage:
  pip <command> [options]

Commands:
  install           Install packages.
  download          Download packages.
  uninstall         Uninstall packages.
  freeze            Output installed packages in requirements format.
  list              List installed packages.
  show              Show information about installed packages.
  check             Verify installed packages have compatible dependencies.
  config            Manage local and global configuration.
  search            Search PyPI for packages.
  wheel             Build wheels from your requirements.
  hash              Compute hashes of package archives.
  completion        A helper command used for command completion.
  help              Show help for commands.

General Options:
  -h, --help          Show help.
  --isolated           Run pip in an isolated mode, ignoring environment variables and user configuration.
  -v, --verbose        Give more output. Option is additive, and can be used up to 3 times.
  -V, --version        Show version and exit.
  -q, --quiet          Give less output. Option is additive, and can be used up to 3 times (corresponding to
                        WARNING, ERROR, and CRITICAL logging levels).
  --log <path>        Path to a verbose appending log.
  --proxy <proxy>      Specify a proxy in the form [user:passwd@]proxy.server:port.
  --retries <retries>  Maximum number of retries each connection should attempt (default 5 times).
  --timeout <sec>      Set the socket timeout (default 15 seconds).
  --exists-action <action> Default action when a path already exists: (s)witch, (i)gnore, (w)ipe, (b)ackup,
                        (a)bort).
  --trusted-host <hostname> Mark this host as trusted, even though it does not have valid or any HTTPS.
  --cert <path>        Path to alternate CA bundle.
  --client-cert <path> Path to SSL client certificate, a single file containing the private key and the
                        certificate in PEM format.
  --cache-dir <dir>    Store the cache data in <dir>.
  --no-cache-dir       Disable the cache.
  --disable-pip-version-check Don't periodically check PyPI to determine whether a new version of pip is available for
                        download. Implied with --no-index.
  --no-color           Suppress colored output
```

Opciones generales de pip

```
pip install requests
pip install beautifulsoup4
pip install simplekml
```

Ejemplo de como instalar librerías en la consola de Python



Paquetes

Los paquetes son una manera de estructurar los espacios de nombres de Python usando “nombres de módulos con puntos”. Por ejemplo, el nombre de módulo A.B designa un sub-módulo llamado B en un paquete llamado A. Tal como el uso de módulos evita que los autores de diferentes módulos tengan que preocuparse de los respectivos nombres de variables globales, el uso de nombres de módulos con puntos evita que los autores de paquetes de muchos módulos, como NumPy o la Biblioteca de Imágenes de Python (Python Imaging Library, o PIL), tengan que preocuparse de los respectivos nombres de módulos.

Suponte que quieres designar una colección de módulos (un “paquete”) para el manejo uniforme de archivos y datos de sonidos. Hay diferentes formatos de archivos de sonido (normalmente reconocidos por su extensión, por ejemplo: .wav, .aiff, .au), por lo que tienes que crear y mantener una colección siempre creciente de módulos para la conversión entre los distintos formatos de archivos. Hay muchas operaciones diferentes que quizás quieras ejecutar en los datos de sonido (como mezclarlos, añadir eco, aplicar una función ecualizadora, crear un efecto estéreo artificial), por lo que además estarás escribiendo una lista sin fin de módulos para realizar estas operaciones. Aquí hay una posible estructura para tu paquete (expresados en términos de un sistema jerárquico de archivos):

<pre>sound/ __init__.py formats/ __init__.py wavread.py wavwrite.py aiffread.py aiffwrite.py auread.py auwrite.py ... effects/ __init__.py echo.py surround.py reverse.py ... filters/ __init__.py equalizer.py vocoder.py karaoke.py ...</pre>	<p>Paquete superior Inicializa el paquete de sonido Subpaquete para conversiones de formato</p> <p>Subpaquete para efectos de sonido</p> <p>Subpaquete para filtros</p>
---	---

Los archivos `__init__.py` se necesitan para hacer que Python trate los directorios como que contienen paquetes; esto se hace para prevenir directorios con un nombre común, como string, de esconder sin intención a módulos válidos que se suceden luego en el camino de búsqueda de módulos. En el caso más simple, `__init__.py` puede ser solamente un archivo vacío, pero también puede ejecutar código de inicialización para el paquete o configurar la variable `__all__`, descrita luego.

Es posible solo utilizar una característica de todo el paquete:

```
import sound.effects.echo
```



Archivos

En Python, así como en cualquier otro lenguaje, los archivos se manipulan en tres pasos: primero se abren, luego se opera sobre ellos y por último se cierran.

Apertura

Para abrir un archivo debemos usar la función `open()`, que recibe como parámetros el nombre del archivo y el modo en el que se debe abrir. De forma predeterminada (es decir, si se omite el segundo parámetro), el archivo se abre como sólo lectura.

Es importante tener en cuenta que todas las operaciones están limitadas a la forma en la que se abra el archivo: no se puede leer de un archivo abierto solamente para escritura, ni escribir en un archivo abierto como solo lectura.

Modos

- `r`: Sólo lectura. No se podrá escribir en el archivo.
- `w`: Sólo escritura. Trunca el archivo al momento de abrirlo. `a`: Sólo escritura. Escribe al final del archivo.

En cualquiera de los modos, si el archivo no existe, es creado. Opcionalmente se puede añadir `+` al modo para que se abra en modo lectura y escritura a la vez; aunque esto no suele ser necesario y requiere cuidado para que funcione correctamente.

Otro modificador posible es `b`, que sirve para trabajar con archivos binarios. Esto es necesario en Windows para manejar correctamente archivos de imágenes, o música (toda clase de archivos que no sean texto simple), porque el mismo SO hace diferencia entre archivos binarios y de texto. Esto no sucede en sistemas tipo UNIX (como Mac OS, o Linux), y por tanto en estos sistemas el modificador `b` no hace ninguna diferencia.

Lectura

Una vez abierto el archivo, podemos leer el contenido hacia una cadena con `read()`, leer una línea con `readline()`, u obtener una lista conteniendo las líneas del archivo con `readlines()`. Los tres métodos aceptan un parámetro entero opcional que define el número máximo de bytes a leer del archivo. Si este parámetro es negativo o simplemente se omite, `read` y `readlines` leerán todo el archivo y `readline` una línea completa sin importar su largo.

Otra forma de leer el archivo es leer línea por línea en un bucle `for`, ya que el objeto archivo es iterable.



1.readline().

Leerá el texto desde la posición en que se localice hasta encontrar el caracter de escape retorno de línea (\n).

2.readlines().

Leerá el texto desde la posición en que se localice y creará un objeto de tipo tuple que contenga cada línea dentro del archivo.

3.writelines().

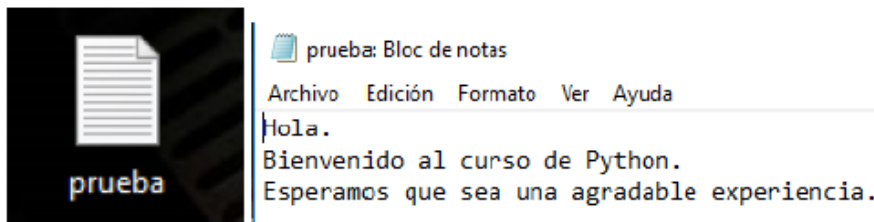
Escribirá el texto contenido dentro de un elemento de tipo list o tuple.

Ejemplo:

- Se creará un archivo de texto nuevo con el nombre prueba.txt.
- Al objeto se le asignará el nombre archivo.
- Se escribirán 3 líneas.
- Se desplegará la posición del puntero del archivo.
- Se desplegará el tipo de dato que es la variable archivo.
- Se cerrará el archivo.

```
1 archivo = open("prueba.txt", "w")
2 archivo.write("Hola.\nBienvenido al curso de Python.\nEsperamos
que sea una agradable experiencia.")
3 print(archivo.tell())
4 print(type(archivo))
5 archivo.close()
```

Al finalizar código, compruebe que el archivo fue creado en el escritorio





Ejemplo:

- Se abrirá el archivo prueba.txt como sólo lectura.
- Al objeto se le asignará el nombre archivo.
- Se leerá la primera línea de texto.
- Se desplegará dicha línea.
- Se cerrará el archivo.

```
C:\Users\joel.guerrero\Desktop>py lectura.py  
Hola.
```

```
1 archivo = open("prueba.txt", "r")  
2 print(archivo.readline())  
3 archivo.close()
```

Ejemplo:

- Se abrirá el archivo nuevo.txt como sólo lectura.
- Al objeto se le asignará el nombre archivo.
- Se localizará el puntero en la posición 12 del archivo.
- Se leerán todas las líneas de texto a partir de dicha posición.
- Se desplegará cada línea.
- Se cerrará el archivo.

```
C:\Users\joel.guerrero\Desktop>py lectura.py  
Bienvenido al curso de Python.  
  
Esperamos que sea una agradable experiencia.
```

```
1 archivo = open("prueba.txt", "r")  
2 archivo.seek(12)  
3 for linea in archivo.readlines():  
4     print(linea)  
5 archivo.close()
```



Advertencia

Si un archivo existente se abre en modo lectura-escritura, al escribir en él se sobrescribirán los datos anteriores, a menos que se haya llegado al final del archivo.

Este proceso de sobre escritura se realiza carácter por carácter, sin consideraciones adicionales para los caracteres de fin de línea ni otros caracteres especiales.

Escritura

Si lo que queremos es escribir en el archivo, tenemos los métodos `write` y `writelines`. Contrapartes de `read` y `readlines` respectivamente, `write` escribe una cadena al archivo y `writelines` recibe una lista de líneas para escribir. Por ejemplo, si quisiéramos recrear el archivo `prueba.txt` del ejemplo anterior, podemos hacerlo de dos formas:

Ejemplo de escritura

```
1 archivo = open('prueba.txt', 'w') # escritura y truncado
2 archivo.write("Esto es\nuna prueba\nde lectura!")
3 archivo.writelines(['Esto es\n', 'una prueba\n', 'de lectura!'])
4
5 # notar la inclusión explícita de saltos de línea
6 archivo.close()
```

prueba: Bloc de notas

Archivo Edición Formato Ver Ayuda

Esto es
una prueba
de lectura!Esto es
una prueba
de lectura!

```
C:\Users\joel.guerrero\Desktop>py lectura.py
C:\Users\joel.guerrero\Desktop>
```

Cierre de archivos:

Cuando terminamos de trabajar con el archivo, lo cerramos con `close()`. Esto libera el archivo para ser usado por otros programas, y además asegura que los cambios sobre él se guarden. De más está decir que Python se encarga de cerrar todos los archivos que queden abiertos al final del programa, pero es una buena práctica no dejar nada al azar y cerrar el archivo tan pronto como se lo termina de usar.



Bases de Datos

¿Qué es una base de datos?

Una base de datos es un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso. En este sentido; una biblioteca puede considerarse una base de datos compuesta en su mayoría por documentos y textos impresos en papel e indexados para su consulta. Actualmente, y debido al desarrollo tecnológico de campos como la informática y la electrónica, la mayoría de las bases de datos están en formato digital, siendo este un componente electrónico, por tanto, se ha desarrollado y se ofrece un amplio rango de soluciones al problema del almacenamiento de datos.

Existen programas denominados sistemas gestores de bases de datos, abreviado SGBD (del inglés Database Management System o DBMS), que permiten almacenar y posteriormente acceder a los datos de forma rápida y estructurada.

Herramienta a utilizar en la sesión

MySQL Workbench es una herramienta visual unificada para arquitectos de bases de datos, desarrolladores y DBA. MySQL Workbench proporciona modelado de datos, desarrollo de SQL y herramientas de administración integrales para la configuración del servidor, la administración de usuarios, las copias de respaldo y mucho más. MySQL Workbench está disponible en Windows, Linux y Mac OS X.

Conceptos de Base de Datos

En bases de datos, una tabla es el lugar donde se almacenan datos, que tienen características similares y conforman la estructura de la base de datos. Los datos que pertenecen a un mismo elemento se organizan en columnas o campos. Estos tienen datos del mismo tipo. Se llama registro, a cada una de las filas o tuplas, que forman una tabla.

Beneficios para crear tablas en MySQL

En función de las necesidades que tengas puedes crear diversos tipos de tablas en MySQL. Pongamos unos ejemplos para entenderlo mejor:

Mayor velocidad

En el caso de necesitar una mayor velocidad tienes la posibilidad de cargar las tablas en la RAM del servidor, de esta manera se aprovecharía mucho más todo el potencial.

Mayor espacio:

Si lo que necesitas es tener más espacio tendrás que ahorrarlo de alguna manera. Para hacerlo MySQL tiene que comprimir las tablas. Otra opción, es que simplemente queramos tener un sistema para almacenar datos, sin más, y nos de igual tanto nivel de detalle.



Crear una tabla en MySQL

Empezaremos con la codificación de cómo crear una tabla dentro de la opción MySQL Workbench, en su editor de código

```
CREATE TABLE nombre_tabla
```

Con el código superior tenemos la sentencia estándar para crear la tabla, solamente tenemos que poner el nombre de la tabla, nombre de la columna y su tipo.

Podemos tener una base de datos con numerosas tablas, por lo que al crear una nueva podemos tener una existente con el mismo nombre. Para evitar problemas debemos usar la sentencia 'IF NOT EXISTS':

```
1 CREATE TABLE IF NOT EXISTS nombre_tabla
2 (definición de la tabla,
3 definición de columnas,
4 tipos de columnas
5 );
```

También podemos crear bases de datos en la consola de comandos incluida en el programa MySQL Workbench, accediendo a ella desde el menú inicio.

Al dar clic sobre la opción, podremos ver una ventana como esta:

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2053
Server version: 5.6.24 Source distribution

Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> █
```

Para crear la base de datos "alumnos" introducimos el siguiente comando:

```
CREATE DATABASE alumnos;
```

La consola nos responderá lo siguiente:

```
[mysql> CREATE DATABASE alumnos;
Query OK, 1 row affected (0,00 sec)
```



Como seleccionar la base de datos con la consola de MySQL

Esto es algo sencillo pero necesario.

Para que mysql sepa a qué base de datos nos vamos a referir cuando introduzcamos los comandos, debemos indicárselo mediante el comando:

USE nombre_tabla

Por consiguiente, para el ejemplo que nos ocupa introducimos el siguiente comando para usar la base de datos "alumnos":

```
USE alumnos;
```

Crear una tabla en la base de datos.

Ahora crearemos una tabla porque sin tablas, la base de datos no nos sirve de mucho la verdad.

Para este ejemplo voy a crear una tabla muy sencilla. Mi tabla alumnos sólo tendrá dos campos: Nombre y edad, para empezar a familiarizarnos. Entonces, siguiendo un modelo sencillo, estos serían los campos que voy a crear en la tabla:

nombre -> VARCHAR(30)

edad -> INT

Ahora que ya lo tenemos claro vamos a crear la tabla con el comando "CREATE TABLE" de la siguiente forma:

```
CREATE TABLE alumnos (nombre VARCHAR(30), edad INT);
```

Comprobar la tabla creada.

Este paso no es necesario, pero si aconsejable ya que salimos de dudas en cuanto a que se ha creado nuestra tabla, si es que teníamos alguna duda.

Para comprobar los campos de la tabla que acabamos de crear podemos utilizar el comando DESCRIBE con el nombre de la tabla que queremos comprobar. Por lo tanto, para nuestro ejemplo ejecutamos el comando

```
DESCRIBE alumnos;
```

```
[mysql> DESCRIBE alumnos;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| nombre | varchar(30)   | YES  |     | NULL    |       |
| edad   | int(11)       | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.03 sec)
```

Resultado en la consola de comandos



Añadir algunos registros.

Para terminar, vamos a ver cómo podemos añadir contenido a nuestra base de datos desde consola.

Antes de ejecutar los comandos, pongo a continuación los datos que voy a introducir:

- Jose Sanchez 22
- Alberto Jurado 19
- Carlos Martin 20

Para añadir registros a la base de datos usaremos el comando INSERT INTO seguido del nombre de la tabla y posteriormente VALUES seguido de los valores que queremos incluir.

Sin más, este sería el comando para introducir los registros solicitados:

```
INSERT INTO alumnos VALUES ('Jose Sanchez', 22);
```

```
INSERT INTO alumnos VALUES ('Alberto Jurado', 19);
```

```
INSERT INTO alumnos VALUES ('Carlos Martín', 20);
```

La consola de comandos nos muestra lo siguiente

```
[mysql> INSERT INTO alumnos VALUES ('Jose Sanchez', 22);  
Query OK, 1 row affected (0,01 sec)
```

Comprobar los datos.

Y para terminar simplemente vamos a comprobar que la base de datos o, mejor dicho, la tabla alumnos tiene los datos que acabamos de introducir. Para mostrar los datos de una tabla podemos usar el comando SELECT FROM para que nos muestre todos los registros (*) de la tabla:

```
SELECT * FROM alumnos;
```

Resultado en consola de comandos:

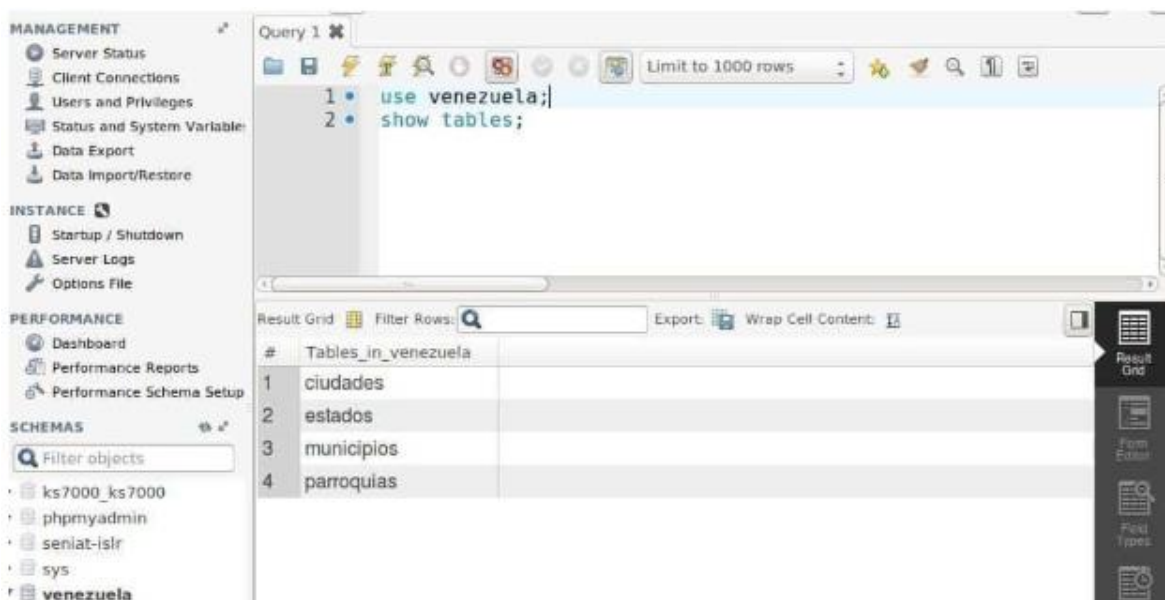



```
[mysql> SELECT * FROM alumnos;
```

nombre	edad
Jose Sanchez	22
Alberto Jurado	19
Carlos Martín	20

```
3 rows in set (0,00 sec)
```

Los comandos descritos anteriormente, aplican cuando se abre el editor de My SQL Workbench



Para ejecutar el código para crear una base de datos, insertar filas o columnas dentro de la misma, tenemos un editor de código, el cual debe ser ejecutado en la opción , el cual ejecutara el código, y le indicara si fue añadido satisfactoriamente.

En la parte izquierda de la ventana, en la sección Schemas, se encuentran las bases de datos que hemos creado, aparte de las agregadas en el sistema.

Diseño de bases de datos

CREAR LA CONEXIÓN A LA BASE DE DATOS

Para que puedas crear una base de datos primero creas una conexión al servidor de MySQL, para esto entras a MySQL Workbench.

Lo que estás haciendo en esta imagen es crear una conexión, con la finalidad de poder interactuar con el servidor de base de datos (crear bases de datos, vistas, comandos SQL etc.)

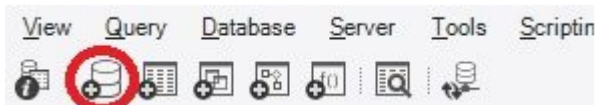
Las partes más importantes son:

1. El nombre del servidor=127.0.0.1 o que también puede ser localhost.
2. El puerto=3306
3. El nombre de usuario=root.

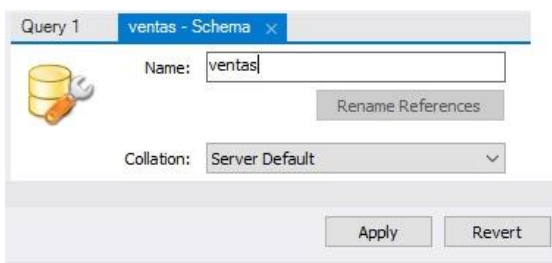
Una vez creada la conexión te aparecerá en la vista principal de MySQL Workbench.

Lo siguiente que tienes hacer es entrar dando doble click, te va pedir la clave del usuario con el que creaste la conexión.

A continuación, debes crear la base de datos, para esto sigue los pasos de la imagen:



Le das el nombre como quieres que se llame y aplicas los cambios



O puedes crearla por medio de su consola de comandos, usando el comando CREATE

Ejemplo:

```
CREATE database uanl;
```

Ejecutas la instrucción, y aparecerá creada la base de datos, también observa en cierta parte de la ventana, la exitosa creación de la misma.

CREAR EL MODELO

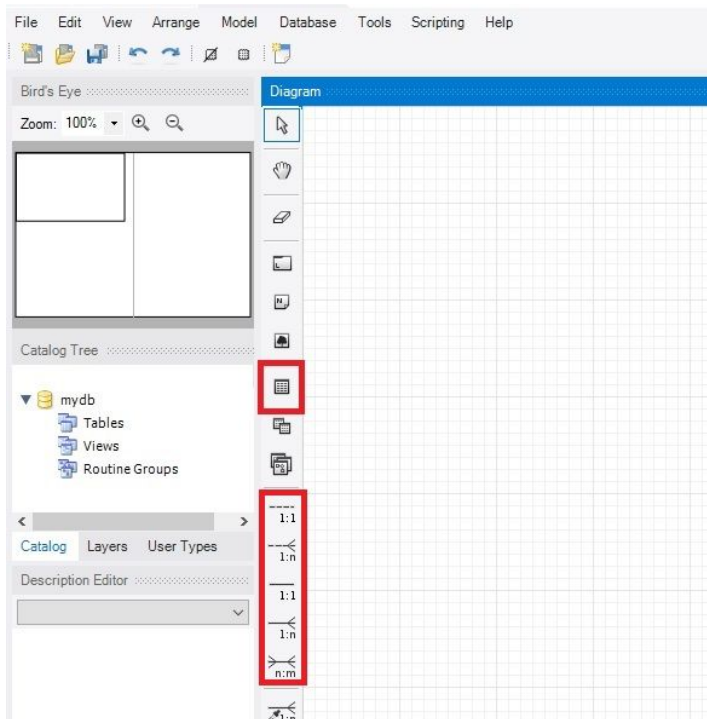
Un modelo es un script en el que podemos crear todo el diseño de las tablas y relaciones de una base de datos pero de forma gráfica, se los puede importar a nuestra base de datos y automáticamente se crean todo lo creado en el modelo.

Para crear el modelo debes entrar a MySQL Workbench, y sigues el paso que se ve en la imagen.



Lo siguiente que vas hacer es crear un diagrama, el cual es una especie de lienzo, sobre el que puedes crear todas las tablas de tu base de datos

Para crear el diagrama sigues las indicaciones de la imagen.





A continuación, se muestra un lienzo donde a través de arrastrar y soltar, puedes crear tablas, relaciones y los atributos para cada tabla.

CREAR LA TABLA

Para esto das un click en el primer recuadro rojo que se ve en la imagen anterior y luego das otro click dentro del lienzo, en el área de recuadros pequeños y te aparecerá una tabla por defecto con el nombre table1.

CÓMO AGREGAR CAMPOS A LA TABLA

Para agregar campos debes dar doble click sobre la tabla creada table1.

Seguidamente en la parte inferior te aparece una ventana donde puedes agregar y modificar los atributos de la tabla.

The screenshot shows a window titled 'table1 - Table'. Inside, there's a 'Table Name' field containing 'table1' (highlighted with a red box and labeled '1') and a 'Schema' field containing 'mydb'. Below this is a table with columns: 'Column Name', 'Datatype', 'PK', 'NN', 'UQ', 'B', 'UN', 'ZF', 'AI', 'G', and 'Default/Expression'. A red arrow labeled '2' points to the first empty row in the 'Column Name' column.

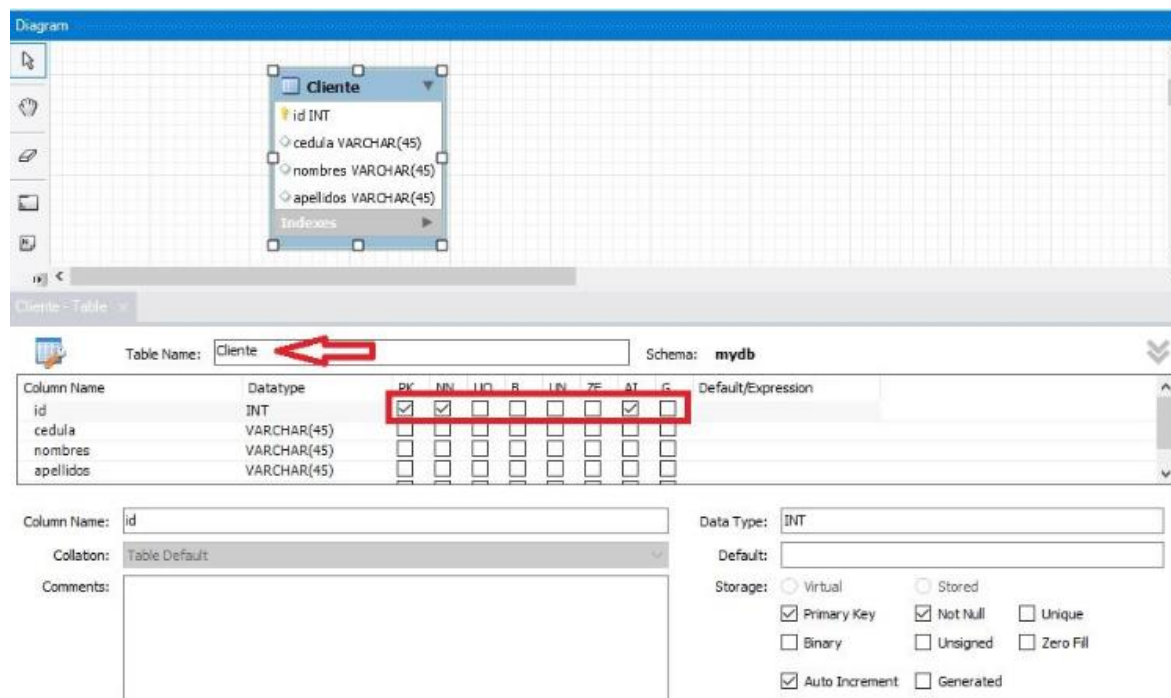
En el primer campo recuadro rojo es para el nombre de tu tabla. Para agregar campos debes dar doble click en área de la flecha número 2.

Esto lo debes repetir de acuerdo al número de campos que quieras ingresar.

En esta parte también le das el tipo de dato que quieres que tenga ese campo (Datatype), así mismo configuras que un campo sea clave primaria (PK), si quieres que el campo no acepte nulos (NN), si deseas que un campo sea autonumérico (AI).

Recuerda que por ejemplo si deseas que un campo se autonumérico (AI), debes dejar checado el campo AI.

En la siguiente imagen, se muestra un ejemplo gráfico de una tabla con campos, en una base de datos:



Como te puedes dar cuenta en el campo id de la tabla le he puesto que sea clave primaria (PK), que no contengan valores null (NN) y que sea un campo auto incrementable (AI).

CÓMO CREAR RELACIONES ENTRE TABLAS

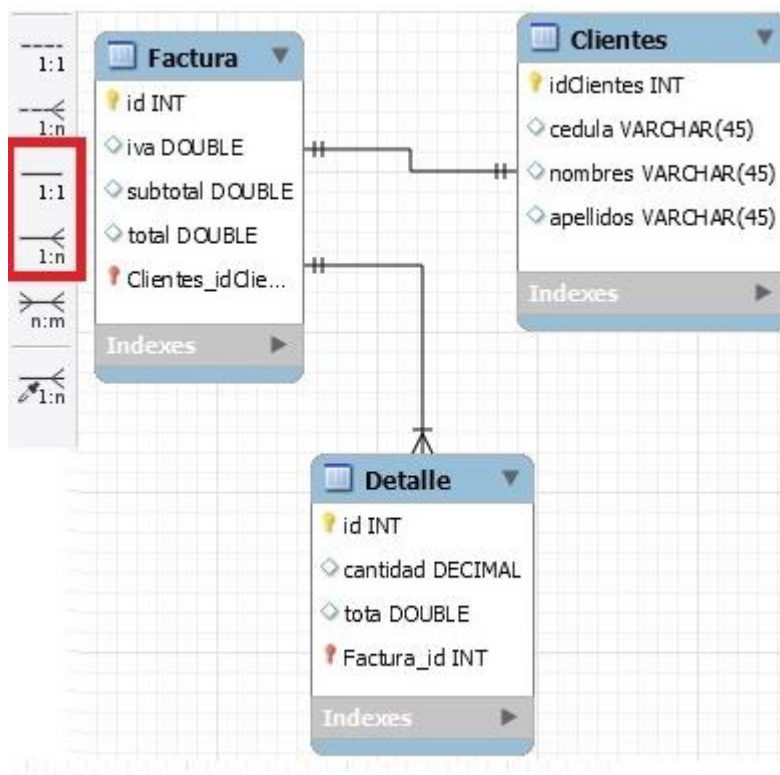
Además de crear tablas también podemos crear relaciones entre ellas, por ejemplo, una típica relación maestro detalle que es una relación de uno a muchos y por ejemplo se la puede implementar en una factura.

Para crear una relación das click en la relación que desees (uno a uno, uno a muchos etc.) y primero das click en la tabla que quieres que se cree la clave foránea y luego en la tabla desde donde quieres que se herede la clave primaria.

En la imagen posterior de ejemplo: la tabla Detalle tiene la clave foránea de la tabla Factura.

Entonces el proceso de crear la relación es elegir el tipo de relación y luego dar click en la tabla Detalle y finalmente en la tabla Factura

De la misma forma para la relación Cliente–Factura que es de uno a uno, en este caso quiero que la clave foránea se cree en la tabla Factura.

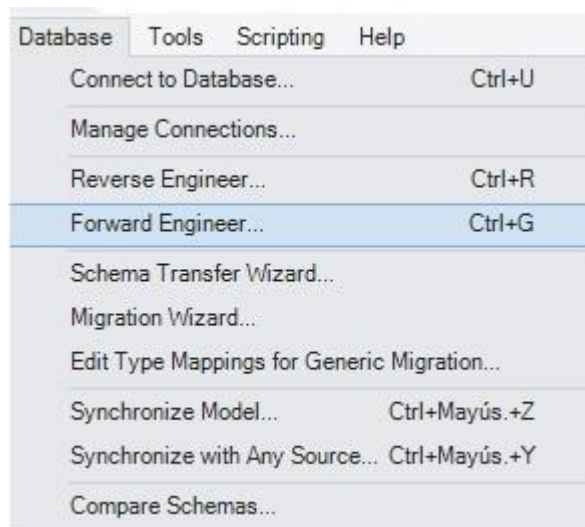


CÓMO CREAR LAS TABLAS EN LA BASE DE DATOS A PARTIR DEL MODELO

Ahora bien, el modelo lo vamos a pasar a la base de datos de manera que se generen las tablas con sus respectivos campos y las relaciones.

A esto se le conoce como Forward Engineer.

Desde el modelo sigues los pasos, como se muestra en la imagen:





Esta parte hace referencia a la conexión y a sus atributos, por lo general se quedan los que están por defecto.

The screenshot shows the 'Forward Engineer to Database' window with the 'Set Parameters for Connecting to a DBMS' tab selected. The left sidebar contains 'Connection Options' with sub-items: 'Options', 'Select Objects', 'Review SQL Script', and 'Commit Progress'. The main area has the following fields:

- Stored Connection: (dropdown menu) with a hint 'Select from saved connection settings'.
- Connection Method: (dropdown menu) with a hint 'Method to use to connect to the RDBMS'.
- Parameters tab is active, showing:
 - Hostname: Port: (Hint: Name or IP address of the server host - and TCP/IP port.)
 - Username: (Hint: Name of the user to connect with.)
 - Password: (Hint: The user's password. Will be requested later not set.)
 - Default Schema: (Hint: The schema to use as default schema. Leave blank to select it later.)

At the bottom right are buttons: 'Back', 'Next' (highlighted with a red box), and 'Cancel'.

The screenshot shows the 'Forward Engineer to Database' window with the 'Set Options for Database to be Created' tab selected. The left sidebar is the same as the previous screenshot. The main area has the following sections:

- Tables:
 - ☐ Skip creation of FOREIGN KEYS
 - ☐ Skip creation of FK Indexes as well
 - ☐ Generate separate CREATE INDEX statements
 - ☐ Generate INSERT statements for tables
 - ☐ Disable FK checks for INSERTs
- Other Objects:
 - ☐ Don't create view placeholder tables
 - ☐ Do not create users. Only create privileges (GRANTS)
- Code Generation:
 - ☐ DROP objects before each CREATE object
 - ☐ Generate DROP SCHEMA
 - ☐ Omit schema qualifier in object names
 - ☐ Generate USE statements
 - ☐ Add SHOW WARNINGS after every DDL statement
 - ☒ Include model attached scripts

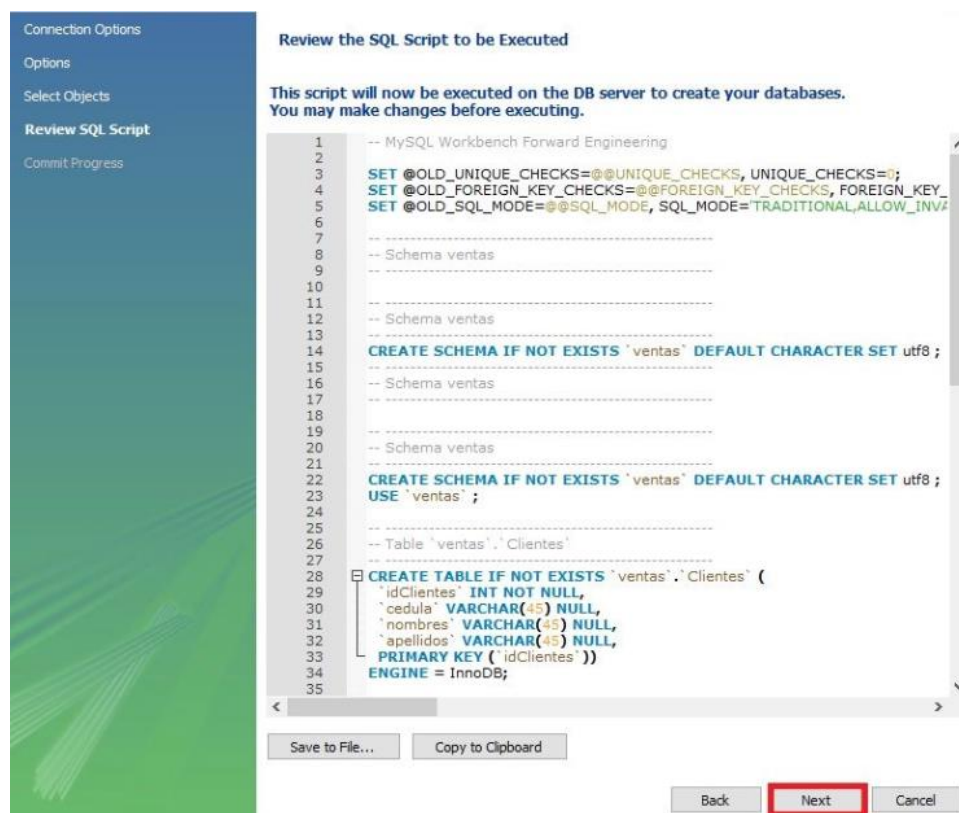
At the bottom right are buttons: 'Back', 'Next' (highlighted with a red box), and 'Cancel'.



Aquí te muestra todos lo que se creará en la base de datos, lo principal que vemos en la parte superior de la imagen, es que son 3 tablas las que se van a crear.



Ahora vas ejecutar el script interno que se genera detrás del modelo.



Te muestra que todo ha ido perfecto y que se han generado de forma correcta las tablas, sus relaciones y pues básicamente con esto termina.





Lo que queda es revisar que se crearon las tablas junto con sus relaciones.





Comandos Básicos MySQL

Definiendo cómo es almacenada la información.

- CREATE DATABASE se utiliza para crear una nueva base de datos vacía.
- DROP DATABASE se utiliza para eliminar completamente una base de datos existente.
- CREATE TABLE se utiliza para crear una nueva tabla, donde la información se almacena realmente.
- ALTER TABLE se utiliza para modificar una tabla ya existente.
- DROP TABLE se utiliza para eliminar por completo una tabla existente.

Manipulando los datos.

- SELECT se utiliza cuando quieres leer (o seleccionar) tus datos.
- INSERT se utiliza cuando quieres añadir (o insertar) nuevos datos.
- UPDATE se utiliza cuando quieres cambiar (o actualizar) datos existentes.
- DELETE se utiliza cuando quieres eliminar (o borrar) datos existentes.
- REPLACE se utiliza cuando quieres añadir o cambiar (o reemplazar) datos nuevos o ya existentes.
- TRUNCATE se utiliza cuando quieres vaciar (o borrar) todos los datos de la plantilla.

Otros comandos de utilidad:

- Devolver las columnas y la información de la columna correspondiente a la tabla designada:
`show columns from [table name];`
- Mostrar ciertas filas seleccionadas con el valor "lo que sea":
`SELECT * FROM [table name] WHERE [field name] = "whatever";`
- Mostrar todos los registros que contengan el nombre "x" AND el número de teléfono '3444444':
`SELECT * FROM [table name] WHERE name = "x" AND phone_number = '3444444';`
- Mostrar todos los registros que contienen el nombre "x" AND el número de teléfono '3444444' ordenados por el campo "phone_number":
`SELECT * FROM [table name] WHERE name != "x" AND phone_number = '3444444' order by phone_number;`



- Mostrar todos los registros que comienzan con la palabra 'X' y el número de teléfono '3444444':
`SELECT * FROM [table name] WHERE name like "x%" AND phone_number = '3444444';`
- Usar una expresión regular para encontrar registros. Usar "REGEXP BINARY" para forzar la sensibilidad a las mayúsculas. Esto encuentra cualquier registro que comience con "a":
`SELECT * FROM [table name] WHERE rec RLIKE "^a$";`
- Mostrar registros únicos:
`SELECT DISTINCT [column name] FROM [table name];`
- Mostrar los registros seleccionados, ordenados en orden ascendente (asc) o descendente (desc):
`SELECT [col1],[col2] FROM [table name] ORDER BY [col2] DESC;`
- Devolver un número de filas:
`SELECT COUNT(*) FROM [table name];`
- Sumar el contenido de la columna:
`SELECT SUM(*) FROM [table name];`
- Unir tablas en columnas comunes:
`select lookup.illustrationid, lookup.personid,person.birthday from lookup left join person on lookup.personid=person.personid=statement to join birthday in person table with primary illustration id;`
- Cambiar a mysql db. Crear un nuevo usuario:
`INSERT INTO [table name] (Host,User>Password)
VALUES('%','user',PASSWORD('password'));`
- Cambiar la contraseña de un usuario (desde el prompt de MySQL):
`SET PASSWORD FOR 'user'@'hostname' = PASSWORD('passwordhere');`
- Permitir que el usuario "x" se conecte con el servidor de localhost usando la contraseña "passwd":
`grant usage on *.* to x@localhost identified by 'passwd';`
- Cambiar a mysql db. Otorgar privilegios de usuario para una db: `INSERT INTO [table name] (Host,Db,User,Select_priv,Insert_priv,Update_priv>Delete_priv>Create_priv,Drop_priv)
VALUES ('%','databasename','username','Y','Y','Y','Y','Y','N');`

Otra alternativa sería: `grant all privileges on databasename.* to username@localhost;`
- Actualizar datos en una tabla:



UPDATE [table name] SET Select_priv = 'Y', Insert_priv = 'Y', Update_priv = 'Y' where [field name] = 'user';

- Eliminar filas de una tabla:
DELETE from [table name] where [field name] = 'whatever';
- Actualizar base de datos permissions/privileges:
FLUSH PRIVILEGES;
- Eliminar una columna:
alter table [table name] drop column [column name];
- Agregar una nueva columna de una base de datos:
alter table [table name] add column [new column name] varchar (20);
- Cambiar el nombre de una columna:
alter table [table name] change [old column name] [new column name] varchar (50);
- Crear una columna única para no tener inconvenientes:
alter table [table name] add unique ([column name]);
- Hacer más grande una columna:
alter table [table name] modify [column name] VARCHAR(3);
- Eliminar una columna única de una tabla:
alter table [table name] drop index [colmn name];
- Cargar un archivo CSV dentro de una tabla:
LOAD DATA INFILE '/tmp/filename.csv' replace INTO TABLE [table name] FIELDS
TERMINATED BY ',' LINES TERMINATED BY 'n' (field1,field2,field3);
- Volcar todas las bases de datos en una copia de seguridad. El backup posee comandos sql para recrear todos las db:
[mysql dir]/bin/mysqldump -u root -ppassword --opt >/tmp/alldatabases.sql
- Volcar una base de datos para el back up:
[mysql dir]/bin/mysqldump -u username -ppassword --databases databasename
>/tmp/databasename.sql
- Volver una tabla de una base de datos:
[mysql dir]/bin/mysqldump -c -u username -ppassword databasename tablename >
>/tmp/databasename.tablename.sql



- Restaurar una base de datos (o tabla) de un backup:
[mysql dir]/bin/mysql -u username -ppassword databasename < /tmp/databasename.sql

Ejemplo de cómo se usarían los comandos de MySQL para crear una base de datos con tablas e información:

```
CREATE DATABASE mydb;
USE mydb;
CREATE TABLE mitabla ( id INT PRIMARY KEY, nombre VARCHAR(20) );
INSERT INTO mitabla VALUES ( 1, 'Will' );
INSERT INTO mitabla VALUES ( 2, 'Marry' );
INSERT INTO mitabla VALUES ( 3, 'Dean' );
SELECT id, nombre FROM mitabla WHERE id = 1;
UPDATE mitabla SET nombre = 'Willy' WHERE id = 1;
SELECT id, nombre FROM mitabla;
DELETE FROM mitabla WHERE id = 1;
SELECT id, nombre FROM mitabla;
DROP DATABASE mydb;
SELECT count(1) from mitabla; da el número de registros en la tabla
```

Ejemplo de clase



Ejercicios para practicar. Puede crear las tablas como tú lo desees en MySQL workbench

Crear una tabla1:

```
CREATE TABLE [table name] (firstname VARCHAR(20), middleinitial VARCHAR(3), lastname  
VARCHAR(35),suffix VARCHAR(3),officeid VARCHAR(10),userid VARCHAR(15),username  
VARCHAR(8),email VARCHAR(35),phone VARCHAR(25), groups  
VARCHAR(15),datestamp DATE,timestamp time,pgpemail VARCHAR(255));
```

Crear tabla2:

```
create table [table name] (personid int(50) not null auto_increment primary key,firstname  
varchar(35),middlename varchar(50),lastnamevarchar(50) default 'bato');
```



Conexión de bases de datos con Python

No abordaremos aquí, conocimientos básicos sobre MySQL, uso, instalación ni configuración. Sin perjuicio de lo anterior, haré una breve introducción.

Structured Query Language (SQL)

SQL es el lenguaje de consulta estructurado utilizado para el acceso a bases de datos relacionales. Si bien SQL como lenguaje, posee ciertos estándares, el lenguaje de consulta en sí, varía para cada base de datos en particular, siendo el tratado en este ejemplo, el correspondiente a MySQL.

Introducción a bases de datos con Python

En el caso particular de Python, el acceso a bases de datos se encuentra definido a modo de estándar en las especificaciones de DB-API (por curiosidad, puedes visitar Python Database API specification).

Esto significa, que para utilizar cualquier base de datos, siempre se deberán seguir los mismos pasos:

Importar el módulo de conexión (en nuestro caso, utilizaremos PyMySQL)

```
import pymysql
```

Conectarse a la base de datos

Vea el siguiente ejemplo:

```
import pymysql

def GetAlumnos():
    #CONFIGURACION
    conn =
    pymysql.connect(host='localhost',
    port=3306, user='root',
    passwd='delarosa123', db='uanl')

    #CREAMOS NUEVO CURSOR
    cursor = conn.cursor()
    #EXECUTO QUERY
    cursor.execute("select * from
    alumnos")

    #OBTENGO EL RESULTADO
    alumnos = cursor.fetchall()

    cursor.close()
    conn.close()

    result = ""
    for alumno in alumnos:
        result = result + "<tr>"
        for columna in alumno:
            result = result + "<td>" +
            str(columna) + "</td>"
            result = result + "</tr>"

    return result
```

12:25 p. m.



Comencemos por observar el código

En la base de datos recién creada en workbench, siempre debemos llevar la siguiente nomenclatura:

```
c = pymysql.connect(host='localhost', user="root", db="agenda")
```

- 1.- En la conexión debemos poner el host de la base de datos, en este caso localhost
- 2.- si hay puertos de por medio, poner una función port, indicando el puerto a seguir en la base de datos
- 3.- Colocar el usuario de la base de datos
- 4.- En el caso de la contraseña, ponerla. Si no tiene, dejar el espacio vacío, pero sin dejar de colocar comillas sencillas
- 5.- Indicar la base de datos a la que queremos acceder

Ya conectados a la base de datos solo queda comenzar con las consultas, pymysql lo hace todo mediante cursores, veamos como:

```
#CREAMOS NUEVO CURSOR
cursor = conn.cursor()
#EXECUTO QUERY
cursor.execute("select * from
alumnos")
```

Veamos otro ejemplo, para que quede más claro:

```
try:
    with c.cursor() as cursor:
        sql = "INSERT INTO `persona` (`nombre`, `apellido`, `email`)
VALUES(%s, %s, %s)"
        cursor.execute(sql, ("hernan", "castilla",
"hcastillag@gmail.com"))
        c.commit()
finally:
    pass
```

Cuando se escribe el cursor.execute() y se trata de un insert, la consulta solo es almacenada en la conexión, para poder ejecutarla tenemos que hacer un c.commit(), esto se encargara de guardar toda la información en la base de datos.



Veamos otro ejemplo

```
import pymysql

##### CONFIGURAR ESTO #####
# Abre conexion con la base de datos
db = pymysql.connect("database_host","username","password","database_name")
#####

# prepare a cursor object using cursor() method
cursor = db.cursor()

# ejecuta el SQL query usando el metodo execute().
cursor.execute("SELECT VERSION()")

# procesa una unica linea usando el metodo fetchone().
data = cursor.fetchone()
print ("Database version : {0}".format(data))

# desconecta del servidor
db.close()
```

Este ejemplo nos permite ver la versión de la base de datos que estamos utilizando



Insertar filas en la base de datos

Bien, ahora que sabemos que todo funciona correctamente, es hora de comenzar a guardar datos.

El comando es INSERT INTO, seguido del nombre de la tabla y de la palabra VALUES que, entre paréntesis, contiene una lista con todos los campos de que consta el registro. Como siempre, no olvides el punto y coma final.

Atiende cuidadosamente: todos los campos de que consta el registro y, además, en el mismo orden en que aparecen en la estructura de la tabla.

Ya veremos qué hay más formas de introducir datos, pero primero debes manejarte bien con esta.

Véase el siguiente ejemplo:

```
import pymysql

##### CONFIGURAR ESTO #####
# Open database connection
db = pymysql.connect("database_host","username","password","database_name")
#####

# prepare a cursor object using cursor() method
cursor = db.cursor()

# Prepare SQL query to INSERT a record into the database.
sql = "INSERT INTO test(id, name, email) \
VALUES (NULL, '{0}', '{1}')" .format("cosme", "testmail@sever.com")
try:
    # Execute the SQL command
    cursor.execute(sql)
    # Commit your changes in the database
    db.commit()
except:
    # Rollback in case there is any error
    db.rollback()

# desconectar del servidor
db.close()
```

Hay algunas cosas que debes tener en cuenta:

La tabla es sensible a las mayúsculas, por lo que debemos usar test y no TEST

Para evitar escribir la columna índice, debes pasar NULL. Es decir, que esa columna siempre tiene que tener algún valor, no puede estar vacía.



Al final agregar una instrucción que cierre la conexión, al terminar de agregar en la base de datos.

Si corres este código, vas a notar que una nueva fila fue agregada a tu base de datos.

Veamos otro ejemplo:

```
import pymysql

# Conectar con base de datos
conexion = pymysql.connect(host="localhost",
                           user="alejandro",
                           passwd="2018_alejandro",
                           database="personal")

cursor = conexion.cursor()

# Agregar nueva tabla 'Oficinas' a la base de datos 'personal'
TablaOficinas = """CREATE TABLE Oficinas(
denom CHAR(20),
provin CHAR(10),
PRIMARY KEY (denom))"""

cursor.execute(TablaOficinas)
print("Se ha agregado la tabla 'Oficinas' a la base de datos")

# Cerrar conexión
conexion.close()
```

Cuando agregamos una tabla, siempre hay que especificar los tipos de valores de las columnas de la tabla, y poner la llave primaria que se va a utilizar en la base de datos.

En algunos casos, se pone auto_increment o autoincrementable.

El not null se agrega para especificar que una columna no acepta el valor NULL, es decir, que esa columna siempre tiene que tener algún valor, no puede estar vacía.



Leer base de datos

Este caso es muy similar al de INSERT. Vamos a ver un simple ejemplo con SELECT, pero puedes usar cualquier comando de SQL.

```
import pymysql

##### CONFIGURAR ESTO #####
# Open database connection
db = pymysql.connect("database_host","username","password","database_name")
#####

# prepare a cursor object using cursor() method
cursor = db.cursor()

# Prepare SQL query to READ a record into the database.
sql = "SELECT * FROM test \
WHERE id > {0}".format(0)

# Execute the SQL command
cursor.execute(sql)

# Fetch all the rows in a list of lists.
results = cursor.fetchall()
for row in results:
    id = row[0]
    name = row[1]
    email = row[2]
    # Now print fetched result
    print ("id = {0}, name = {1}, email = {1}".format(id,name,email))

# disconnect from server
db.close()
```

Asegúrate de tener algo escrito en tu tabla o el código no devolverá nada.



Veamos otro ejemplo sencillo de Consulta o lectura de base de datos

```
try:
    with c.cursor() as cursor:
        sql = "SELECT * FROM persona"
        cursor.execute(sql)
        query = cursor.fetchall()
        print(query)
finally:
    pass
```

curso.fetchall() nos retorna todas las filas contenidas, la información de las filas se encuentra almacenada en tuplas, el resultado obtenido es el siguiente:

((('hernan', 'castilla', 'hcastilla@gmail.com'),)

Mostrar tablas y obtener datos de la tabla nueva



```
import pymysql

# Conectar con base de datos
conexion = pymysql.connect(host="localhost",
                           user="alejandro",
                           passwd="2018_alejandro",
                           database="personal")

cursor = conexion.cursor()

# Recuperar registros de la tabla 'Oficinas'
tablas = "SHOW TABLES;"
registros = "SELECT * FROM Oficinas;"

# Mostrar tablas
cursor.execute(tablas)
filas = cursor.fetchall()
print("Tablas de 'personal':")
for fila in filas:
    print(fila)

# Mostrar registros
cursor.execute(registros)
filas = cursor.fetchall()
print("Registros de 'Oficinas':")
for fila in filas:
    print(fila[0], ":", fila[1])

# Finalizar
conexion.commit()
conexion.close()
```



Eliminar datos en la base de datos con Python

Por último, un ejemplo para borrar la tabla Oficinas de la base de datos.

```
import pymysql

# Conectar con base de datos
conexion = pymysql.connect(host="localhost",
                           user="alejandro",
                           passwd="2018_alejandro",
                           database="personal")

cursor = conexion.cursor()

# Construir comando para borrar tabla
BorrarTabla = "DROP TABLE IF EXISTS Oficinas;"

# Borrar tabla
cursor.execute(BorrarTabla)
print("Se ha suprimido una tabla de la base de datos")

# Finalizar transacción y cerrar
conexion.commit()
conexion.close()
```



Introducción a HTML

HTML, que significa Lenguaje de Marcado para Hipertextos (HyperText Markup Language) es el elemento de construcción más básico de una página web y se usa para crear y representar visualmente una página web. Determina el contenido de la página web, pero no su funcionalidad. Otras tecnologías distintas de HTML son usadas generalmente para describir la apariencia/presentación de una página web (CSS) o su funcionalidad (JavaScript).

Hiper Texto se refiere a enlaces que conectan una página Web con otra, ya sea dentro de una página web o entre diferentes sitios web. los vínculos son un aspecto fundamental de la Web. Al subir contenido a Internet y vincularlo a páginas de otras personas, te haces participante activo de esta Red Mundial.

Los documentos HTML son archivos de texto plano (también conocidos como ASCII) que pueden ser creados mediante cualquier editor de texto, aunque también existen programas específicos para editar HTML, concebidos específicamente para editar páginas web en HTML.

Los archivos HTML tienen la extensión .html ó htm y para ver la estructura de una página web en lenguaje HTML, los navegadores suelen disponer de un menú con la opción "Ver" desde la que se puede visualizar el código fuente de la página HTML. Dicho código fuente nos dará una idea clara de en qué consiste este lenguaje que, como hemos dicho anteriormente, es un simple lenguaje de marcas entre cuyas funciones destaca la posibilidad de enlazar documentos y partes de documentos, esto es, la hipertextualidad.

Etiquetas

HTML incluye el diseño gráfico de las páginas web, sin embargo, eso no es cierto ya que HTML sólo sirve para indicar como va ordenado el contenido de una página web. Esto lo hace por medio de las marcas de hipertexto las cuales son **etiquetas** conocidas en inglés como tags.

Hasta el momento ya tenemos una idea sobre lo que hace HTML pero aún no sabemos cómo funcionan las etiquetas. Cabe mencionar que las etiquetas no sólo sirven para ordenar nuestro contenido, sino que ayudan a los buscadores a encontrar la información por medio de las etiquetas.

Ahora que entendemos el concepto de etiquetas, necesitamos poder identificarlas. Éstas se caracterizan porque van dentro de los caracteres menor que < y mayor que >, como en este ejemplo: <Aquí dentro va el nombre de la etiqueta>. Aunque eso no es todo debido a que deben tener un inicio y un fin. El inicio de una etiqueta es normal, en otras palabras, lleva los dos caracteres que mencioné anteriormente y dentro de estos se encuentra el nombre, sin embargo, existe una diferencia para el fin de la etiqueta ya que antes de escribir el nombre de la etiqueta debemos escribir una diagonal /. Como en este ejemplo:

```
<Inicio de la etiqueta>  
</Fin de la etiqueta>
```



Existen ciertas excepciones con el fin de las etiquetas, ya que algunas usualmente sólo se utilizan con el inicio de la etiqueta, ejemplos de esto son las etiquetas BR (salto de línea), IMG (etiqueta para poner una imagen), entre otras.

Estructura

La estructura básica de una página web es la siguiente:

```
<html>

<head></head>

<body>

</body>

</html>
```

Una estructura HTML se empieza con la etiqueta <html> y acaba con </html>. Todo lo que esté en medio será la página web. Dentro de <html></html> se encuentran 2 partes diferenciadas. La primera <head></head> es la cabecera de la página. Aquí irán cierta información que no es directamente el contenido de la página. Aquí se pone el título de la página, los metadatos, estilos, código javascript. La primera que se suele estudiar es <title></title>, que indica el título de la página.

La segunda parte es <body></body>. Aquí va propiamente el contenido de la página: fotos, párrafos, formularios, etc. Por ejemplo, siguiendo con el ejemplo de la página anterior, el siguiente código, podemos cambiar el título de la página.

Ejemplos de etiquetas que van en <body>:

-
 => Indica salto de línea. En HTML un salto de línea normal (pulsando la tecla Enter) no produce un salto de línea en el resultado. Es necesario escribir
 (u otra etiqueta similar).
- => Indica comienzo y fin de negrita.
- <i></i> => Itálica.
- seis etiquetas de títulos/subtítulos: de "<h1>" hasta "<h6>", correspondiendo los números al nivel jerárquico. Es decir, una etiqueta "<h1>" sería el nivel jerárquico máximo, "<h2>" los subtítulos de los niveles h1 y así sucesivamente.
- Etiqueta í => Esto indica que queremos poner una "i" con acento, es decir, "í".

<body background="fondo.gif">	El documento tendrá como fondo la imagen indicada.
<body bgcolor="white" text="blue" link="red" vlink="red">	El fondo será blanco, el texto azul, y todos los links (visitados o no) serán rojos



Diseño de una página Web

Cómo trabajar con títulos y párrafos de texto en HTML

HTML tiene seis etiquetas de títulos/subtítulos: de “<h1>” hasta “<h6>”, correspondiendo los números al nivel jerárquico.

Es decir, una etiqueta “<h1>” sería el nivel jerárquico máximo, “<h2>” los subtítulos de los niveles h1 y así sucesivamente.

Por otra parte, los párrafos de texto normal se marcan con “<p>”.

Con ambas cosas tenemos los elementos básicos para crear texto en HTML.

Si comparamos esto con Word, por ejemplo, las etiquetas de títulos serían el equivalente a los estilos “Título 1”, “Título 2”, etc. y la etiqueta de párrafo el equivalente al estilo “normal”. De hecho, al pegar un texto desde Word a un editor HTML visual, así se deberían convertir en el código HTML generado.

Cómo crear listas en páginas web

Las listas son otro elemento indispensable en un contenido. Es muy fácil crearlas en HTML, en el caso de las listas enumeradas se usa la etiqueta y en el caso de las no enumeradas (con viñetas), se usa la etiqueta . En ambos casos se usa para los elementos de la lista.

Por ejemplo:

```
<ol>
  <li>El primer punto</li>
  <li>El segundo punto</li>
  <li>El tercer punto</li>
</ol>
```

Fíjate, además, que, siendo una lista enumerada, no vienen los números, es decir, el “1.) ...”, “2.) ...”, etc. Eso es así porque, de manera similar a Word, usar listas enumeradas implica precisamente que los números se crean automáticamente cuando se visualice la página en un navegador web.

Cómo usar tablas en páginas web

Las tablas son un elemento ya algo más avanzado y, por tanto, aquí me limitará a una estructura básica de tabla:

Aquí, la etiqueta `<table>`, como ya te puedes imaginar, demarca la tabla.

Dentro de ella, cualquier fila de cualquier tipo se demarca con `<tr>` que viene de “table-row” (fila de tabla en inglés). Dentro de esto, puede haber diferentes tipos de filas que se diferencian por el tipo de celdas que contienen.

En este ejemplo, tenemos una primera fila con celdas con la etiqueta `<th>` (“table header”, cabecera de tabla en inglés) que actúan como cabecera y luego el resto, serían celdas ordinarias con datos, `<td>`, “table data”.

```
1 <table>
2   <tr>
3     <th>Nombre</th>
4     <th>Apellido</th>
5     <th>Web</th>
6   </tr>
7   <tr>
8     <td>Amy</td>
9     <td>Porterfield</td>
10    <td>https://www.amyporterfield.com/</td>
11  </tr>
12  <tr>
13    <td>Pat</td>
14    <td>Flynn</td>
15    <td>https://www.smartpassiveincome.com/</td>
16  </tr>
17 </table>
```

Insertar enlaces (links) en páginas HTML

Vamos ahora a lo que es la etiqueta por excelencia en una página web: la etiqueta de un enlace.

Esta etiqueta es “`<a>`”, de “anchor” (ancla en inglés) y su sintaxis básica es la siguiente:

```
1 <a target="_blank" href="[url del enlace]">Texto del enlace</a>
```

La información clave, el enlace, viene en el atributo href, aquí debes insertar la URL del sitio que quiere enlazar.

El atributo target también es muy importante. Sus valores más usados (y los únicos que necesitas realmente en la práctica) son “_blank”, que quiere decir la página del enlace se abre en una nueva pestaña/ventana, y “self”, que la abre en la pestaña/ventana del documento actual.

Si omites este atributo o dejas el valor en blanco, se asume por defecto target = “self”.



Insertar imágenes en una página HTML

La etiqueta HTML para insertar una imagen en HTML es `` y su sintaxis básica la siguiente:

Aquí el atributo `src` es el equivalente a `href` en el caso del enlace anterior, es decir, una URL que apunta a la imagen que quieres visualizar.

```
1 
```

Por ejemplo: `http://midominio.com/imagenes/mi-imagen.png`

Otro atributo muy importante es `alt`. Por una parte, porque en caso de que el enlace a la imagen esté roto, visualizará un texto alternativo (de aquí, el nombre de “`alt`” del atributo) que será el texto asociado a este atributo y, por otra parte, porque Google lo tiene muy en cuenta para posicionar la imagen (en la búsqueda de imágenes).

Los atributos `height` y `width` te permiten especificar las dimensiones a las que ajustar la imagen en el navegador. De no ser esas las dimensiones originales de la imagen, el navegador reducirá o ampliará la imagen, según corresponda, para ajustarla a estas dimensiones.

Cómo poner color al texto en HTML

Ahora ya que lo fácil: ponerle color a un elemento HTML.

De manera general, puedes especificar el color de cualquier elemento HTML con el atributo `style`, con la misma sintaxis usada en este ejemplo:

```
1 <h2 style="color=rgb(13,44,84)">Título H2 en un azul oscuro</h2>
```

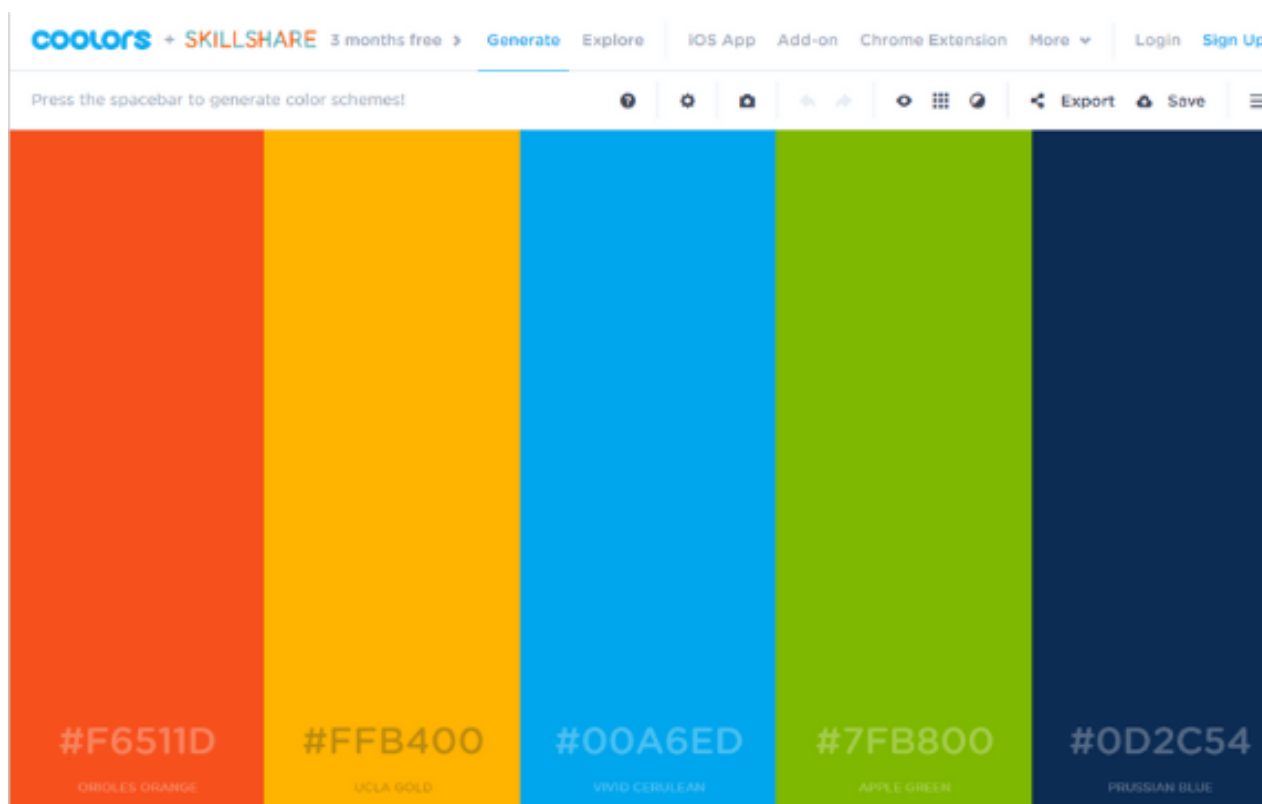
Del mismo modo puedes usar también valores hexadecimales:

```
1 <h2 style="color=#0D2C54">Título H2 en un azul oscuro</h2>
```

Cómo poner un color de fondo en HTML

Y esto mismo lo puedes usar también para especificar el color de fondo, en este caso aplicando a atributo anterior a la etiqueta `body`:

```
1 <body style="color=#FFB400">
```



Color	HTML / CSS Name	Hex Code #RRGGBB	Decimal Code (R,G,B)
	Black	#000000	(0,0,0)
	White	#FFFFFF	(255,255,255)
	Red	#FF0000	(255,0,0)
	Lime	#00FF00	(0,255,0)
	Blue	#0000FF	(0,0,255)
	Yellow	#FFFF00	(255,255,0)
	Cyan / Aqua	#00FFFF	(0,255,255)
	Magenta / Fuchsia	#FF00FF	(255,0,255)
	Silver	#C0C0C0	(192,192,192)
	Gray	#808080	(128,128,128)
	Maroon	#800000	(128,0,0)
	Olive	#808000	(128,128,0)
	Green	#008000	(0,128,0)
	Purple	#800080	(128,0,128)
	Teal	#008080	(0,128,128)
	Navy	#000080	(0,0,128)

Colores básicos y su codificación RGB en decimal y hexadecimal.



Programación Web con Django

Django, en este caso un marco de desarrollo de código abierto en Python especialmente útil para la creación rápida de aplicaciones web.

Características de Django:

- El objetivo esencial de este marco de desarrollo es la creación de aplicaciones web sin complicaciones. Pretende ser sencillo, rápido, como ellos mismos afirman, “puedes concentrarte en escribir tu aplicación sin la necesidad de tener que reinventar la rueda”. Es un resumen casi perfecto de Django: programación rápida de páginas y aplicaciones web.
- Basado en la filosofía DRY (Don't Repeat Yourself: No te repitas). Muchas aplicaciones web y proyectos digitales comparten numerosas líneas de código unos con otros. Django es el marco de desarrollo de refactorización de código casi por excelencia. Permite reutilizar programación de unas aplicaciones a otras sin la obligación de tener que repetir las mismas líneas de código entre distintos proyectos.
- Django es un framework web de alto nivel basado en el paradigma Modelo-Vista-Controlador. No podría ser de otra forma en un marco de desarrollo que apuesta por la sencillez, la rapidez y la reutilización de código. Por tanto, por un lado están los datos (el modelo) y por otro la interfaz de usuario (la vista) y la lógica de negocio (el controlador).
- Base de datos embebida. Django viene con SQLite, una base de datos usada por compañías tan importantes como Facebook o Bloomberg.
- Sistema ORM de conexión a bases de datos. ORM es el sistema por el que este marco de desarrollo se conecta y almacena sus datos en la BD. Dentro de este proceso son esenciales los Querysets, listados de datos del modelo que pueden ser leídos, ordenados y filtrados. En este pequeño tutorial se explica cómo escoger, ordenar o filtrar objetos del modelo.
- Una API propia para el desarrollo de proyectos digitales. Django REST framework es un conjunto de herramientas que permite a los desarrolladores la construcción de Web APIs navegables, compatibles con lectura de datos ORM o no ORM y con políticas de autenticación basadas en paquetes tan conocidos como OAuth1 y OAuth2.

Instalación del módulo de Django

Primeros pasos para desarrollar con Django

La descarga del archivo comprimido con la carpeta de Django y su instalación en cualquier máquina, ya sea Windows, Mac OSX o Linux, es muy sencilla. Para empezar a desarrollar con Django habría que seguir los siguientes pasos:

- Descarga del archivo comprimido.

Se debe comenzar la instalación de Django en el equipo mediante el uso de pip. Comando:

```
pip install Django
```

Django tiene algunas dependencias y requisitos que también deben estar instalados en la máquina si se quiere usar este marco:

- La primera de esas dependencias es el lenguaje de programación Python. Lo más recomendable es utilizar la versión Python 3, aunque Django también es compatible con la versión 2.7.
- La segunda es la instalación de un servidor web independiente (Django dispone de su propio servidor de desarrollo ligero, que no puede ser usado como un servidor de producción, sino solo para desarrollo). Este servidor debe estar basado en la especificación WSGI PEP 3333. Esta especificación permite elegir una amplia selección de servidores para proyectos digitales con Django.
- El tercer requisito es una base de datos compatible. Hay varias opciones: PostgreSQL, MySQL, SQLite3 y Oracle.

Instalar Django con pip

```
C:\Users\joel.guerrero>pip install django
Collecting django
  Downloading https://files.pythonhosted.org/packages/c7/87/fbd666c4f87591ae25b7bb374298e8629816e87193c4099d3608ef11fab9/Django-2.1.7-py3-none-any.whl (7.3MB)
    100% |#####| 7.3MB 3.7MB/s
Collecting pytz (from django)
  Downloading https://files.pythonhosted.org/packages/61/28/1d3920e4d1d50b19bc5d24398a7cd85cc7b9a75a490570d5a30c57622d34/pytz-2018.9-py2.py3-none-any.whl (510kB)
    100% |#####| 512kB 7.9MB/s
Installing collected packages: pytz, django
```



Ya instalado Django, y los archivos necesarios, comprobemos que se han instalado en la carpeta raíz

Este equipo > Windows (C:) > Usuarios > joel.guerrerop

Nombre	Fecha de modifica...	Tipo	Tamaño
.AndroidStudio3.1	06/11/2018 04:10 ...	Carpeta de archivos	
.AndroidStudio3.2	06/11/2018 05:51 ...	Carpeta de archivos	
.gradle	06/11/2018 04:33 ...	Carpeta de archivos	
.nbi	11/09/2018 06:26 ...	Carpeta de archivos	
.VirtualBox	14/02/2019 11:23 a...	Carpeta de archivos	
AndroidStudioProjects	06/11/2018 05:55 ...	Carpeta de archivos	
AppData	22/05/2018 08:08 a...	Carpeta de archivos	
Búsquedas	13/12/2018 05:54 ...	Carpeta de archivos	
Contactos	13/12/2018 05:54 ...	Carpeta de archivos	
Descargas	21/02/2019 08:39 a...	Carpeta de archivos	
Desktop	21/02/2019 09:19 a...	Carpeta de archivos	
Documents	15/02/2019 06:40 ...	Carpeta de archivos	
Favoritos	13/12/2018 05:54 ...	Carpeta de archivos	
HP	19/05/2018 05:12 ...	Carpeta de archivos	
Imágenes	09/01/2019 12:52 ...	Carpeta de archivos	
IntelGraphicsProfiles	21/02/2019 08:36 a...	Carpeta de archivos	
Juegos guardados	13/12/2018 05:54 ...	Carpeta de archivos	
MicrosoftEdgeBackups	02/09/2018 12:37 ...	Carpeta de archivos	
Música	13/12/2018 05:54 ...	Carpeta de archivos	
mysite	20/02/2019 10:28 a...	Carpeta de archivos	
Objetos 3D	13/12/2018 05:54 ...	Carpeta de archivos	
OneDrive	27/08/2018 01:02 ...	Carpeta de archivos	
Videos	13/12/2018 05:54 ...	Carpeta de archivos	
Vínculos	13/12/2018 05:54 ...	Carpeta de archivos	
VirtualBox VMs	14/02/2019 09:36 a...	Carpeta de archivos	

La carpeta que se descarga se llama mysite, y predeterminadamente se instala en la carpeta c:/usuarios/nombre_de_usuario.

Si esta es la primera vez que utiliza Django, tendrá que hacerse cargo de ciertas configuraciones iniciales. Concretamente, tendrá que autogenerar un código que establezca un Django project – un conjunto de ajustes para una instancia de Django, incluida la configuración de la base de datos, opciones específicas de Django y configuraciones específicas de la aplicación.

Desde la línea de comandos, cambie a un directorio donde le gustaría almacenar su código, luego, ejecute el siguiente comando:

```
...> django-admin startproject mysite
```

En algunos casos, le dirá que ya existe la carpeta de mysite. Así debe quedar en la consola de comandos de Windows.

```
C:\Users\joel.guerrerop>django-admin startproject mysite
CommandError: 'C:\Users\joel.guerrerop\mysite' already exists
```



Vaya a la ruta especificada, para comprobar los archivos ya creados

```
mysite/
  manage.py
  mysite/
    __init__.py
    settings.py
    urls.py
    wsgi.py
```

Usuarios > joel.guerrero > mysite > mysite

Buscar en mysite

Nombre	Fecha de modifica...	Tipo	Tamaño
__init__	19/02/2019 10:08 a...	Python File	0 KB
settings	19/02/2019 10:08 a...	Python File	4 KB
urls	19/02/2019 10:08 a...	Python File	1 KB
wsgi	19/02/2019 10:08 a...	Python File	1 KB

Estos archivos son:

- El directorio raíz externo mysite/ solo es un contenedor de su proyecto. Su nombre no es relevante para Django; usted puede cambiarle el nombre a lo que quiera.
- manage.py: Una utilidad de la línea de comandos que le permite interactuar con este proyecto Django de diferentes formas.
- En interior del directorio mysite/ es el propio paquete de Python para su proyecto. Su nombre es el nombre del paquete de Python que usted tendrá que utilizar para importar todo dentro de este (por ejemplo, mysite.urls).
- mysite/__init__.py: Un archivo vacío que le indica a Python que este directorio debería ser considerado como un paquete Python.
- mysite/settings.py: Ajustes/configuración para este proyecto Django. Django settings le indicará todo sobre cómo funciona la configuración.
- mysite/urls.py: Las declaraciones URL para este proyecto Django; una «tabla de contenidos» de su sitio basado en Django. Puede leer más sobre las URLs en URL dispatcher.
- mysite/wsgi.py: Un punto de entrada para que los servidores web compatibles con WSGI puedan servir su proyecto.

El servidor de desarrollo

Comprobemos que su proyecto Django funciona. Cambie al directorio externo mysite, si todavía no lo ha hecho, y ejecute los siguientes comandos:

```
...\> py manage.py runserver
```

```
C:\Users\joel.guerrero\mysite>py manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).

You have 15 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin,
auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
February 20, 2019 - 10:28:49
Django version 2.1.7, using settings 'mysite.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Usted ha iniciado el servidor en desarrollo de Django, un servidor web ligero escrito puramente en Python. Se incluye con Django para que pueda desarrollar cosas rápidamente sin tener que lidiar con la configuración de un servidor en producción, como Apache, hasta que esté listo para la producción.

Ahora es un buen momento para tener en cuenta que: no debe utilizar este servidor en algo parecido a un entorno de producción. Está pensado sólo para usarse durante el desarrollo (nuestro trabajo es crear frameworks Web, no servidores web.). Ahora que el servidor está funcionando, visite <http://127.0.0.1:8000/> con su navegador Web. Verá la página «Felicitaciones!», con un cohete despegando. ¡Funcionó!

127.0.0.1:8000

django

[View release](#)



The install worked successfully! Congratulations!

You are seeing this page because `DEBUG=True` is in your settings file and you have not configured any URLs.

Puntos Importantes

Cambiando el puerto

De forma predeterminada, el comando runserver inicia el servidor de desarrollo en la IP interna en el puerto 8000.

Si desea cambiar el puerto del servidor, pásalo como un argumento de la línea de comandos. Por ejemplo, este comando inicia el servidor en el puerto 8080:

```
... \> py manage.py runserver 8080
```

Si desea cambiar la IP del servidor, pásela junto con el puerto. Por ejemplo, para escuchar en todas las IPs públicas (útil si usted está ejecutando Vagrant o quiere mostrar su trabajo en otros equipos de la red), utilice:

```
... \> py manage.py runserver 0:8000
```

0 es un atajo para 0.0.0.0. La documentación completa de el servidor de desarrollo se encuentra en la referencia de runserver.

Recarga automática del comando runserver

El servidor de desarrollo recarga de forma automática el código Python para cada petición cuando sea necesario. No es necesario reiniciar el servidor para que los cambios de código surtan efecto. Sin embargo, algunas acciones como la adición de archivos no provocan un reinicio, por lo que tendrá que reiniciar el servidor en estos casos.

```
[20/Feb/2019 10:30:27] "GET / HTTP/1.1" 200 16348
[20/Feb/2019 10:30:27] "GET /static/admin/css/fonts.css HTTP/1.1" 200 423
[20/Feb/2019 10:30:27] "GET /static/admin/fonts/Roboto-Bold-webfont.woff HTTP/1.1" 200 82564
[20/Feb/2019 10:30:27] "GET /static/admin/fonts/Roboto-Regular-webfont.woff HTTP/1.1" 200 80304
[20/Feb/2019 10:30:27] "GET /static/admin/fonts/Roboto-Light-webfont.woff HTTP/1.1" 200 81348
Not Found: /favicon.ico
[20/Feb/2019 10:30:27] "GET /favicon.ico HTTP/1.1" 404 1972
```

Ejercicio

Ahora que su entorno, un «proyecto», se ha configurado, ya está listo para empezar a trabajar.

Cada aplicación que usted escribe en Django consiste en un paquete de Python que sigue una determinada convención. Django tiene una utilidad que genera automáticamente la estructura básica de directorios de una aplicación, por lo que usted puede centrarse en la escritura de código en lugar de crear directorios.

Sus aplicaciones se pueden ubicar en cualquier parte de su :ref:`ruta Python <tut-searchpath>`. En este tutorial vamos a crear nuestra aplicación encuesta junto al archivo `manage.py` para que pueda ser importado como su propio módulo de nivel superior, en lugar de un submódulo de `mysite`.

Para crear su aplicación, asegúrese de que está en el mismo directorio que el archivo `manage.py` y escriba este comando:

```
py manage.py startapp polls
```

Eso va a crear un directorio `polls` que se presenta de la siguiente forma: Visto desde la carpeta raíz:

```
polls/
  __init__.py
  admin.py
  apps.py
  migrations/
    __init__.py
  models.py
  tests.py
  views.py
```



Nombre	Fecha de modifica...	Tipo	Tamaño
mysite	20/02/2019 10:28 a...	Carpeta de archivos	
polls	21/02/2019 09:30 a...	Carpeta de archivos	
db.sqlite3	20/02/2019 10:28 a...	Archivo SQLITE3	0 KB
manage	19/02/2019 10:08 a...	Python File	1 KB

Esta estructura de directorios almacenará la aplicación encuesta.

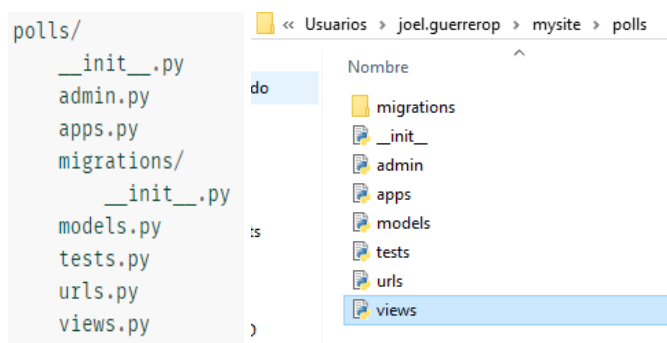
Escriba su primera vista

Vamos a escribir la primera vista. Abra el archivo `polls/views.py`, en su editor de texto, y ponga el siguiente código Python en ella:

```
1 from django.http import HttpResponse
2
3
4 def index(request):
5     return HttpResponse("Hello, world. You're at the polls index.")
```

Esta es la vista más simple posible en Django. Para llamar la vista, tenemos que asignarla a una URL y para ello necesitamos una `URLconf`.

Para crear una `URLconf` en el directorio `encuestas`, cree un archivo llamado `urls.py`. en el editor de texto, y guárdelo en el directorio `polls`. El directorio de su aplicación debe verse así:



El siguiente paso es señalar la `URLconf` raíz en el módulo `polls.urls`. En `cmd`, diríjase a la carpeta `mysite`, y abra en el editor el archivo `urls.py`. Añada un `import` para `django.urls.include` e inserte una `include()` en la lista `urlpatterns`, para obtener:



```
1 """mysite URL Configuration
2
3 The `urlpatterns` list routes URLs to views. For more information please see:
4     https://docs.djangoproject.com/en/2.1/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 include, path
18
19 ▾ urlpatterns = [
20     path('polls/', include('polls.urls')),
21     path('admin/', admin.site.urls),
22 ]
```

La función `include()` permite hacer referencia a otros `URLconfs`. Cada vez que Django encuentra `include()` corta cualquier parte de la URL que coincide hasta ese punto y envía la cadena restante a la `URLconf` incluida para seguir el proceso.

La idea detrás de `include()` es facilitar la conexión y ejecución inmediata de las URLs. Dado que las encuestas están en su propia `URLconf` (`polls/urls.py`) se pueden ubicar en «`/polls/`», «`/fun_polls/`», «`/content/polls/`» o en cualquier otra ruta raíz, y la aplicación todavía seguirá funcionando.

Cuándo utilizar `include()`: Siempre debe usar `include()` cuando incluye otros patrones de URL. `admin.site.urls` es la única excepción a esto.

Usted ha enviado una vista `index` al `URLconf`. Comprobemos si funciona, ejecute el siguiente comando:

```
C:\Users\joel.guerrero\mysite>py manage.py runserver
Performing system checks...

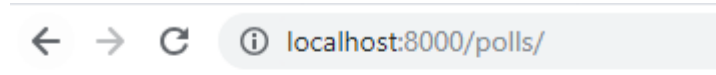
System check identified no issues (0 silenced).

You have 15 unapplied migration(s). Your project may not work
without them. Run 'python manage.py migrate' to apply them.
February 21, 2019 - 09:44:42
Django version 2.1.7, using settings 'mysite.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

```
py manage.py runserver
```



Dirijase a la siguiente dirección web, en su explorador, <http://localhost:8000/polls/>, en su navegador, y usted debería ver el texto «Hello, world. You're at the polls index.» el cual definió en la vista index.



Hello, world. You're at the polls index.

La `path()` función recibe cuatro argumentos, dos requeridos `route` y `view`; y dos opcionales `kwargs` y `name`. Este es el momento de revisar para que sirven estos argumentos.

argumento `path()`: `route`

`route` es una cadena que contiene un patrón de URL. Cuando Django procesa una petición comienza por el primer patrón en `urlpatterns` y continua hacia abajo por la lista comparando la URL solicitada con cada patrón hasta encontrar aquel que calza.

Tenga en cuenta que estas expresiones regulares no buscan parámetros GET y POST o el nombre de dominio. Por ejemplo, en una petición a la dirección URL <https://www.example.com/myapp/>, la `URLconf` buscará `myapp/`. En una petición a <https://www.example.com/myapp/?page=3> la `URLconf` también buscará `myapp/`.

argumento `path()`: `view`

Cuando Django encuentra una coincidencia de expresiones regulares llama a la función de la vista especificada con un objeto `HttpRequest` como primer argumento y cualquiera de los valores «capturados» de la ruta como argumentos de palabra clave. Le daremos un ejemplo de esto en un momento.

argumento `path()`: `kwargs`

Los argumentos arbitrarios de palabra clave se pueden pasar en un diccionario a la vista destino. No vamos a utilizar esta funcionalidad de Django en el tutorial.



argumento path(): name

Dar un nombre a su URL le permite referirse a ella de forma inequívoca desde otras partes de Django sobre todo desde las plantillas. Esta potente característica le permite realizar cambios globales en los patrones de URL de su proyecto modificando solo un único archivo.

Cuando se familiarice con el flujo básico de solicitud y respuesta, lea la parte 2 del presente tutorial para empezar a trabajar con la base de datos.



Configuración de la base de datos

Ahora, abra el archivo `settings.py`, en su editor de texto, ubicado en la carpeta `mysite`. Es un módulo normal de Python con variables de nivel de módulo que representan la configuración de Django.

Por defecto la configuración utiliza SQLite. Si tiene poca experiencia con bases de datos o su interés es solo probar Django esta es la opción más fácil. SQLite está incluido en Python por lo que no tendrá que instalar nada más para soportar su base de datos. Sin embargo, al iniciar su primer proyecto real, es posible que desee utilizar una base de datos más potente como PostgreSQL para evitar dolores de cabeza en el futuro al tener que cambiar entre base de datos.

Si desea utilizar otra base de datos, instale los conectores de base de datos apropiados, y cambie las siguientes claves en el ítem `DATABASES` 'default' para que se ajusten a la configuración de conexión de la base de datos:

`ENGINE` – bien sea `'django.db.backends.sqlite3'`, `'django.db.backends.postgresql_psycopg2'`, `'django.db.backends.mysql'`, o `'django.db.backends.oracle'`. Otros `backends` también están disponibles.

`NAME` –el nombre de su base de datos. Si está utilizando SQLite, la base de datos será un archivo en su computadora; en ese caso `NAME` debe ser la ruta absoluta completa, incluyendo el respectivo nombre del archivo. El valor predeterminado, `os.path.join(BASE_DIR, 'db.sqlite3')`, guardará el archivo en el directorio de su proyecto.

Si no está utilizando SQLite como su base de datos, se deben añadir ajustes adicionales tales como `USER`, `PASSWORD`, y `HOST` se deben añadir.

Para bases de datos distintas a SQLite

Si está utilizando una base de datos SQLite, asegúrese de que ha creado ya la base de datos en este punto. Hágalo con el comando «`CREATE DATABASE database_name;`» en la consola interactiva de la base de datos.

Del mismo modo asegúrese de que la base de datos proporcionada en el archivo `mysite/settings.py` tiene permisos de tipo «`create database`». Esto permitirá la creación automática de `test database` que será necesaria en un tutorial posterior.

Si estás usando SQLite, no es necesario crear nada de antemano - el archivo de base de datos se creará automáticamente cuando sea necesario.

Mientras que usted está editando el archivo `mysite/settings.py`, configure `TIME_ZONE` a su zona horaria.

Además, observe que la configuración de `INSTALLED_APPS` se encuentra en la parte superior del archivo. Esta contiene los nombres de todas las aplicaciones Django que están activadas en esta instancia de Django. Las aplicaciones se pueden usar en diversos proyectos y usted puede empaquetar y distribuirlas para que otras personas las puedan utilizar en sus proyectos.

Por defecto, `INSTALLED_APPS` contiene las siguientes aplicaciones y Django viene equipado con todas ellas:

- `django.contrib.admin` - El sitio administrativo. Usted lo utilizará dentro de poco.
- `django.contrib.auth` - Un sistema de autenticación.
- `django.contrib.contenttypes` - Un framework para los tipos de contenido.
- `django.contrib.sessions` - Un framework de sesión.
- `django.contrib.messages` - Un framework de mensajería.
- `django.contrib.staticfiles` - Un framework para la gestión de archivos estáticos.

Estas aplicaciones se incluyen de forma predeterminada como una conveniencia para el caso común. Algunas de estas aplicaciones utilizan al menos una tabla de base de datos, por lo que necesitamos crear las tablas en la base de datos antes de poder utilizarlas. Para ello, ejecute el siguiente comando:

```
py manage.py migrate
```

El comando `migrate` analiza la configuración `INSTALLED_APPS` y crea las tablas de base de datos necesarias según la configuración de base de datos en su archivo `mysite/settings.py` y las migraciones de base de datos distribuidas con la aplicación. Verá un mensaje para cada migración que aplique. Si está interesado, ejecute el cliente de línea de comandos para su base de datos y teclee `\dt` (PostgreSQL), `SHOW TABLES;` (MySQL), `.schema` (SQLite), o `SELECT TABLE_NAME FROM USER_TABLES;` (Oracle) para desplegar las tablas que creó Django.



Creando modelos

A continuación, definiremos sus modelos, sobre todo su estructura de base de datos, con metadatos adicionales.

Filosofía

Un modelo es la fuente única y definitiva de información sobre sus datos. Contiene los campos esenciales y los comportamientos de los datos que usted guarda. Django sigue el Principio DRY. El objetivo es definir el modelo de datos en un solo lugar y derivar cosas de este automáticamente.

Este incluye las migraciones, a diferencia de Ruby On Rails, por ejemplo, las migraciones se derivan totalmente de su archivo de modelos y son esencialmente sólo un historial a través del cual Django puede pasar para actualizar su esquema de base de datos para que coincida con sus modelos actuales.

En nuestra sencilla aplicación encuesta, vamos a crear dos modelos: Question y Choice . Una Question tiene una pregunta y una fecha de publicación. Una Choice tiene dos campos: el texto de la elección y un conteo de votos. Cada Choice se asocia a una Question.

Estos conceptos se representan mediante clases sencillas de Python. Edite el archivo models.py, ubicado en la carpeta polls, de modo que se vea de la siguiente manera:

```
1  from django.db import models
2
3
4  class Question(models.Model):
5      question_text = models.CharField(max_length=200)
6      pub_date = models.DateTimeField('date published')
7
8
9  class Choice(models.Model):
10     question = models.ForeignKey(Question, on_delete=models.CASCADE)
11     choice_text = models.CharField(max_length=200)
12     votes = models.IntegerField(default=0)
```

El código es sencillo. Cada modelo está representado por una clase que subclasifica django.db.models.Model. Cada modelo tiene una serie de variables de clase, cada uno de los cuales representa un campo de la base de datos en el modelo.

Cada campo está representado por una instancia de una clase Field, por ejemplo, CharField para campos de caracteres y DateTimeField para variables de tiempo y fecha. Esto le dice a Django qué tipo de datos cada campo contiene.



El nombre de cada instancia Field (por ejemplo, `question_text` o `pub_date`) es el nombre del campo, en formato adaptado al lenguaje de la máquina. Va a usar este valor en el código Python y su base de datos va a usarlo como el nombre de la columna.

Puede emplear un primer argumento posicional opcional a una Field para designar un nombre legible por humanos. Ese se utiliza en varias partes introspectivas de Django y sirve al mismo tiempo como documentación. Si no se proporciona este campo, Django usará el nombre legible por la máquina. En este ejemplo, sólo hemos definido un nombre legible para `Question.pub_date`. Para el resto de los campos en este modelo, el nombre del campo legible por la máquina servirá como el nombre legible por humanos.

Algunas clases Field precisan argumentos. La clase `CharField`, por ejemplo, requiere que usted le asigne un `max_length`. Esta se utiliza no sólo en el esquema de la base de datos, sino también en la validación como veremos dentro de poco.

Una clase Field también puede tener varios argumentos opcionales; en este caso, le hemos fijado al default el valor de votes en 0.

Por último, tenga en cuenta que una relación se define usando `ForeignKey`. Eso le indica a Django que cada `Choice` se relaciona con un sola `Question`. Django es compatible con todas las relaciones de bases de datos comunes: varias a una, varias a varias y una a una.

Activando los modelos

Ese pequeño fragmento de código de modelo le proporciona a Django mucha información. Con él Django es capaz de:

- Crear un esquema de base de datos para esta aplicación (oraciones CREATE TABLE).
- Crear una API de acceso a la base datos Python para acceder a los objetos Question y Choice.
- Pero primero tenemos que indicarle a nuestro proyecto que la aplicación polls está instalada.

Para incluir la aplicación en nuestro proyecto necesitamos agregar una referencia a su clase de configuración en la configuración INSTALLED_APPS. La clase PollsConfig está en el archivo polls/apps.py por lo que su ruta punteada es 'polls.apps.PollsConfig'. Edite el archivo mysite/settings.py y agregue la ruta punteada a la configuración INSTALLED_APPS. Se verá así:

```
INSTALLED_APPS = [  
    'polls.apps.PollsConfig',  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
]
```

Ahora Django sabe incluir la aplicación polls. Vamos a ejecutar otro comando:

```
C:\Users\joel.guerrero\mysite>py manage.py makemigrations polls  
Migrations for 'polls':  
  polls\migrations\0001_initial.py  
    - Create model Choice  
    - Create model Question  
    - Add field question to choice
```

Al ejecutar makemigrations, usted le indica a Django que ha realizado algunos cambios a sus modelos (en este caso, ha realizado cambios nuevos) y que le gustaría que los guarde como una migración.

Django guarda los cambios en sus modelos como migraciones (y por lo tanto en su esquema de base de datos); son solo archivos en el disco. Usted puede leer la migración para su nuevo modelo si lo desea, es el archivo polls/migrations/0001_initial.py. No se preocupe, no se espera que usted las lea cada vez que Django hace una, sino que están diseñadas para que sean editables en caso de que usted desee modificar manualmente como Django cambia las cosas.



Hay un comando que ejecutará las migraciones y gestionará el esquema de base de datos automáticamente; este se denomina migrate. El comando sqlmigrate recibe nombres de migración y devuelve su SQL:

```
C:\Users\joel.guerrero\mysite>py manage.py makemigrations polls
Migrations for 'polls':
  polls\migrations\0001_initial.py
    - Create model Choice
    - Create model Question
    - Add field question to choice

C:\Users\joel.guerrero\mysite>py manage.py sqlmigrate polls 0001
BEGIN;
--
-- Create model Choice
--
CREATE TABLE "polls_choice" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "choice_text" varchar(200) NOT NULL, "votes" integer NOT NULL);
--
-- Create model Question
--
CREATE TABLE "polls_question" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "question_text" varchar(200) NOT NULL, "pub_date" datetime NOT NULL);
--
-- Add field question to choice
--
ALTER TABLE "polls_choice" RENAME TO "polls_choice_old";
CREATE TABLE "polls_choice" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "choice_text" varchar(200) NOT NULL, "votes" integer NOT NULL, "question_id" integer NOT NULL REFERENCES "polls_question" ("id") DEFERRABLE INITIALLY DEFERRED);
INSERT INTO "polls_choice" ("id", "choice_text", "votes", "question_id") SELECT "id", "choice_text", "votes", NULL FROM "polls_choice_old";
DROP TABLE "polls_choice_old";
CREATE INDEX "polls_choice_question_id_c5b4b260" ON "polls_choice" ("question_id");
COMMIT;
```

Importante

- La salida exacta variará dependiendo de la base de datos que esté utilizando.
- Los nombres de las tablas se generan automáticamente combinando el nombre de la aplicación (polls) y el nombre del modelo en minúscula; question y choice. (Usted puede anular este comportamiento)
- Las claves primarias (IDs) se agregan automáticamente. (Usted también puede anular esto)
- Convencionalmente, Django añade "_id" al nombre del campo de la clave externa (sí, usted también puede anular esto)
- La relación de la clave externa se hace explícita por una restricción ``FOREIGN KEY``. No se preocupe por las partes DEFERRABLE; eso solo le indica a PostgreSQL que no aplique la clave externa hasta el final de la transacción.
- Se adapta a la base de datos que está utilizando, así que los tipos de campos específicos de las bases de datos como auto_increment (MySQL), serial (PostgreSQL) o integer primary key autoincrement (SQLite) se gestionan de forma automática. Lo mismo se aplica para la cita de nombres de campos, por ejemplo, el uso de comillas dobles o comillas simples.
- El comando sqlmigrate en realidad no ejecuta la migración en su base de datos, sólo la imprime en la pantalla para que pueda ver lo que SQL Django piensa que se requiere. Es útil para comprobar lo que Django va a hacer o si usted tiene administradores de bases de datos que requieran scripts SQL para introducir cambios.

Si le interesa, usted también puede ejecutar el comando `python manage.py check`; este revisa cualquier problema en su proyecto sin hacer migraciones o modificar la base de datos.

```
C:\Users\joel.guerrerop\mysite>py manage.py check
System check identified no issues (0 silenced).
```

A continuación, ejecute de nuevo el comando `migrate` para crear esas tablas modelos en su base de datos:

```
C:\Users\joel.guerrerop\mysite>py manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, polls, 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 polls.0001_initial... OK
  Applying sessions.0001_initial... OK
```

El comando `migrate` toma todas las migraciones que no han sido aplicadas (Django rastrea cuales se aplican utilizando una tabla especial en su base de datos llamada `django_migrations`) y las ejecuta contra su base de datos; básicamente, sincronizando los cambios que usted ha realizado a sus modelos con el esquema en la base de datos.

Las migraciones son muy potentes y le permiten modificar sus modelos con el tiempo, a medida que desarrolla su proyecto, sin necesidad de eliminar su base de datos o las tablas y hacer otras nuevas. Este se especializa en la actualización de su base de datos en vivo, sin perder datos. Recuerde la guía de tres pasos para hacer cambios de modelo:

- Cambie sus modelos (en `models.py`).
- Ejecute el comando `python manage.py makemigrations` para crear migraciones para esos cambios
- Ejecute el comando `python manage.py migrate` para aplicar esos cambios a la base de datos.

Jugando con la API

Ahora vayamos al shell interactivo de Python y juguemos con la API gratuita que Django le proporciona. Para llamar el shell de Python, utilice este comando:

```
C:\Users\joel.guerrerop\mysite>py manage.py shell
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 23:09:28) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from polls.models import Choice, Question
>>> Question.objects.all()
<QuerySet []>
>>> from django.utils import timezone
>>> q=Question(question_text='Que hay de nuevo?', pub_date=timezone.now())
>>> q.save()
>>> q.id
1
>>> q.question_text
'Que hay de nuevo?'
>>> q.pub_date
datetime.datetime(2019, 2, 21, 16, 56, 31, 228792, tzinfo=<UTC>)
>>> q.question_text
'Que hay de nuevo?'
>>> q.save()
>>> Question.objects.all()
<QuerySet [<Question: Question object (1)>]>
```

<Question: Question object (1)> no es una representación útil de este objeto. Arreglemos esto modificando el modelo Question (en el archivo polls/models.py) y agregando un metodo `__str__()` a los dos modelos, Question y Choice:

```
1 from django.db import models
2
3 class Question(models.Model):
4     # ...
5     def __str__(self):
6         return self.question_text
7
8 class Choice(models.Model):
9     # ...
10    def __str__(self):
11        return self.choice_text
```

Es importante añadir los métodos `__str__()` a sus modelos, no solo para su conveniencia al lidiar con la línea de comandos interactiva, sino también porque las representaciones de objetos se usan en todo el sitio administrativo generado automáticamente de Django.



Tenga en cuenta que estos son métodos normales de Python. Vamos a añadir un método personalizado, sólo como demostración:

```
1 import datetime
2
3 from django.db import models
4 from django.utils import timezone
5
6
7 class Question(models.Model):
8     # ...
9     def was_published_recently(self):
10         return self.pub_date >= timezone.now() - datetime.timedelta(days=1)
11
12 class Choice(models.Model):
13     # ...
14     def __str__(self):
15         return self.choice_text
```

Tenga en cuenta la adición de `import datetime` y `from django.utils import timezone`, para hacer referencia el módulo estándar de Python `datetime` y las herramientas relacionadas con el huso horario de Django `django.utils.timezone` respectivamente.

```
>>> from polls.models import Choice, Question

# Make sure our __str__() addition worked.
>>> Question.objects.all()
<QuerySet [<Question: What's up?>]>

# Django provides a rich database lookup API that's entirely driven by
# keyword arguments.
>>> Question.objects.filter(id=1)
<QuerySet [<Question: What's up?>]>
>>> Question.objects.filter(question_text__startswith='What')
<QuerySet [<Question: What's up?>]>

# Get the question that was published this year.
>>> from django.utils import timezone
>>> current_year = timezone.now().year
>>> Question.objects.get(pub_date__year=current_year)
<Question: What's up?>

# Request an ID that doesn't exist, this will raise an exception.
>>> Question.objects.get(id=2)
Traceback (most recent call last):
...
DoesNotExist: Question matching query does not exist.

# Lookup by a primary key is the most common case, so Django provides a
# shortcut for primary-key exact lookups.
# The following is identical to Question.objects.get(id=1).
>>> Question.objects.get(pk=1)
<Question: What's up?>
```




```
# Make sure our custom method worked.
>>> q = Question.objects.get(pk=1)
>>> q.was_published_recently()
True

# Give the Question a couple of Choices. The create call constructs a new
# Choice object, does the INSERT statement, adds the choice to the set
# of available choices and returns the new Choice object. Django creates
# a set to hold the "other side" of a ForeignKey relation
# (e.g. a question's choice) which can be accessed via the API.
>>> q = Question.objects.get(pk=1)

# Display any choices from the related object set -- none so far.
>>> q.choice_set.all()
<QuerySet []>

# Create three choices.
>>> q.choice_set.create(choice_text='Not much', votes=0)
<Choice: Not much>
>>> q.choice_set.create(choice_text='The sky', votes=0)
<Choice: The sky>
>>> c = q.choice_set.create(choice_text='Just hacking again', votes=0)

# Choice objects have API access to their related Question objects.
>>> c.question
<Question: What's up?>

# And vice versa: Question objects get access to Choice objects.
>>> q.choice_set.all()
<QuerySet [<Choice: Not much>, <Choice: The sky>, <Choice: Just hacking again>]>
>>> q.choice_set.count()
3

# The API automatically follows relationships as far as you need.
# Use double underscores to separate relationships.
# This works as many levels deep as you want; there's no limit.
# Find all Choices for any question whose pub_date is in this year
# (reusing the 'current_year' variable we created above).
>>> Choice.objects.filter(question__pub_date__year=current_year)
<QuerySet [<Choice: Not much>, <Choice: The sky>, <Choice: Just hacking again>]>

# Let's delete one of the choices. Use delete() for that.
>>> c = q.choice_set.filter(choice_text__startswith='Just hacking')
>>> c.delete()
```



Agregando nuevas vistas

Escribir aplicaciones web pueden ser monótono, ya que repetimos ciertos patrones una y otra vez. Django intenta quitar algo de esa monotonía en el modelo y en la capa de plantillas, pero los desarrolladores web también experimentan ese aburrimiento a nivel de vistas.

Las Visitas genéricas en Django fueron desarrolladas para aliviar ese dolor. Toman ciertas expresiones y patrones comunes encontrados en el desarrollo de vista y lo abstraen de modo que podamos escribir vistas comunes de datos sin tener que escribir demasiado código.

Uno de los cambios más importantes que se han producido, es que las vistas (views) ahora son clases en lugar de ser funciones. Si queremos podemos seguir usando funciones, pero las clases pueden ser muy rápidas y fáciles de usar.

La forma de usarlas es, básicamente, crear nuestra propia vista heredando de una vista genérica y modificando su comportamiento. Tenemos vistas para traer un objeto de la base de datos (DetailView), para listar objetos (ListView), para crear, actualizar o borrar (CreateView, UpdateView y DeleteView) o podemos utilizar una vista completamente vacía de funcionalidad (View). Usando estas vistas, podremos indicar el modelo de datos que queremos utilizar, filtros para las consultas a base de datos, formularios con los que crearemos los objetos, templates con las que mostraremos la vista y cualquier otro aspecto que necesites modificar.

Utilizando las vistas genéricas tendrás casi todos los comportamientos por defecto que necesitas de manera habitual y, además, serás capaz de modificarlas y adaptarlas a tus necesidades específicas. Si necesitas algo más complejo, puedes crearte tu propia vista basada en clases o simplemente crear una vista a la vieja usanza basada en un método.

Django viene con vistas genéricas que pueden hacer lo siguiente:

- Realizar tareas “simples” y comunes: redirigir a una página diferente y renderizar una plantilla dada.
- Mostrar paginas de listado y detalles para un único objeto. Si estuviéramos creando un aplicación para gestionar conferencias tendríamos una vista `talk_list` y también una `registered_user_list` estas vista serían ejemplos de `ListView` (listado). Una simple página de discusión es un ejemplo de lo que llamamos una vista de `DetailView` (detalle).
- Presentar objetos basados en datos en páginas del tipo fecha, año/mes/día y su detalle asociado, y las paginas “nuevas”.
- Los archivos del blog web de Django (<https://www.djangoproject.com/weblog/>) son archivos del tipo año, mes y día y se construyen de esta forma, como sería un típico archivo de un periódico
- Permitir a los usuarios crear, actualizar y borrar objetos -con o sin autorización-.



Ejemplo:

Para agregar una nueva vista, nos situaremos dentro de la ruta :
NOMBRE_DE_LA_APLICACION/urls.py

Y agregaremos nuestra vista:

```
from django.http import HttpResponse
def index(request):
    return HttpResponse("Hola mundo")
def NuevaVista(request):
    return HttpResponse("Nueva vista")
```

En la ruta: NOMBRE_DE_LA_APLICACION/urls.py

Agregamos al arreglo urlpatterns la ruta de nuestra nueva vista:

```
from django.urls import path
from . import views
urlpatterns = [path('', views.index,name='index'),
               path('NuevaVista/', views. NuevaVista,name=' NuevaVista '),]]
```

Corremos el servidor y accedemos a: NombreAplicacion/NuevaVista

...\> py manage.py runserver

El paso siguiente, será crear un controlador. Crearemos un archivo llamados controller.py en la ruta NOMBRE_DE_LA_APLICACION/

Estructura del archivo creado:



```
NOMBRE_DE_LA_APLICACION/  
    __init__.py  
    admin.py  
    apps.py  
    migrations/  
        __init__.py  
    models.py  
    tests.py  
    urls.py  
    views.py  
    controller.py
```

Abriremos el archivo controller.py, y crearemos una función de prueba que nos devuelva un resultado

```
1 def Saludo(nombre, vista):  
2     return "Bienvenido "+ nombre+" a la vista "+ vista
```

Para acceder a nuestro controlador, solo hay que importarlo a nuestro archivo view,

En la ruta: NOMBRE_DE_LA_APLICACION/views.py

Vamos a agregar el import correspondiente:

```
3  
4 from NOMBRE_DE_LA_APLICACION import controller
```

Al tener importado nuestro controlador, ya tenemos acceso a sus funciones

```
1 from django.shortcuts import render  
2 from django.http import HttpResponse  
3  
4 from NOMBRE_DE_LA_APLICACION import controller  
5  
6 def index(request):  
7     saludo = controller.Saludo("Isaac", "Index")  
8     return HttpResponse(saludo)
```



Usando templates en HTML

Para utilizar pagina HTML físicas, tendremos que agregar una carpeta llamada template, en la

ruta : NOMBRE_DE_LA_APLICACION/nombre

Dentro de esta carpeta crearemos otra que se llame como nuestra aplicación:

```
NOMBRE_DE_LA_APLICACION/  
    __init__.py  
    admin.py  
    apps.py  
    migrations/  
        __init__.py  
    models.py  
    tests.py  
    urls.py  
    views.py  
    controller.py  
    templates/
```

Estructura de la carpeta creada

Sobre la carpeta del proyecto, En la ruta: NOMBRE_Del_PROYECTO/settings.py

Agregue las líneas siguientes al final del archivo

```
120 STATICFILES_DIRS = (  
121     os.path.join(BASE_DIR, "static"),  
122 )  
123  
124 STATIC_URL = '/static/'
```

Importante, En la ruta: NOMBRE_DEL_PROYECTO/settings.py, Agregaremos el nombre de nuestra aplicación al array de aplicaciones:

```
30  
31 # Application definition  
32  
33 INSTALLED_APPS = [  
34     'django.contrib.admin',  
35     'django.contrib.auth',  
36     'django.contrib.contenttypes',  
37     'django.contrib.sessions',  
38     'django.contrib.messages',  
39     'django.contrib.staticfiles',  
40     'NOMBRE_DE_LA_APLICACION'
```



En la ruta: NOMBRE_DE_LA_APLICACION/views.py, Vamos a decirle a nuestra vista que regrese un archivo html:

```
6 def index(request):  
7     return render(request, "NOMBRE_DE_LA_APLICACION/index.html", {})
```

Corremos el servidor y accedemos a : NombreAplicacion/Vista

```
...\> py manage.py runserver
```

Mandar información de mi vista-html

Para mandar información(parámetros, arreglos), en nuestra Vista, debemos incluirlos en la función, tal y como si fuera un diccionario

En la ruta: NOMBRE_DE_LA_APLICACION/views.py

```
6 def index(request):  
7     return render(request, "NOMBRE_DE_LA_APLICACION/index.html",  
8         {  
9             "saludo": "Hola"  
10        })
```

Dentro de nuestra pagina HTML, ya tenemos acceso a la información que se mande desde la vista:

Para acceder a la información, solo se manda llamar el nombre del parametro entre doble llaves

```
1 <!DOCTYPE html>  
2 <html>  
3 <head>  
4     <title>Cursos</title>  
5 </head>  
6 <body>  
7     <h2>Mis cursos</h2>  
8     <h3>{{ saludo }}</h3>  
9 </body>  
10 </html>
```

Mis cursos

Hola

Si lo que vamos a mandar es código HTML, este debe ir entre

```
{% autoescape off %}{{ Mi_parametro }}{% endautoescape %}
```




Acceder a base de datos

Para acceder a BD dentro de Django, el procedimiento es variable según el motor, para este ejemplo se usará MySQL con PyMySQL.

Primero accederemos a nuestro modelo En la ruta: NOMBRE_DE_LA_APLICACION/models.py

```
1 from django.db import models
```

Dentro de nuestro modelo agregaremos alguna clase que nos facilite realizar la conexión

```
1 from django.db import models
2 import pymysql
3 class Model(object):
4     def __init__(self):
5         super(Model, self).__init__()
6
7     def getConnection(self):
8         try:
9             return pymysql.connect(host='localhost', port=3306, user='root',
10                                   password='delarosa123', db='CETI')
11         except Exception as e:
12             return None
13
14     def ObtenerCursosActivos(self):
15         try:
16             conexion = self.getConnection()
17             if conexion == None:
18                 return None
19             try:
20                 with conexion.cursor() as cursor:
21                     consulta = "SELECT nombre, costo, facilitador FROM cursos where activo =1"
22                     cursor.execute(consulta)
23                     result = cursor.fetchall()
24                     return result
25             finally:
26                 conexion.close()
27         except Exception as e:
28             return None
```



Ya tenemos nuestra clase MODELO, como usamos MVC, nuestro controlador será el encargado de las funciones de negocio, hay que agregar la referencia al modelo desde nuestro controlador.

Vamos a crear una función que instancie nuestra clase Model y mande llamar la función que nos devolverá un resultado

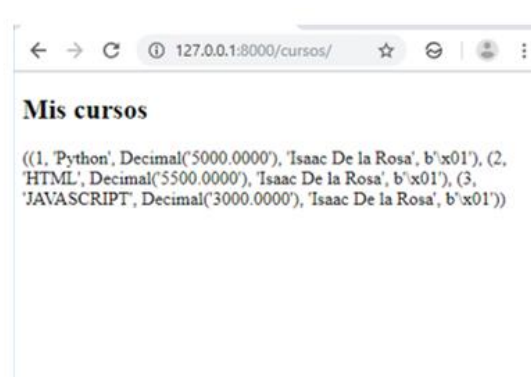
```
1 from NOMBRE_DE_LA_APLICACION import models
2
3 def ObtenerCursos():
4     Model = models.Model()
5     '''Obtenemos los cursos'''
6     result = Model.ObtenerCursosActivos()
7     return result
```

Desde nuestra vista, vamos a mandar lo que nos devuelva el controlador

```
1 from django.shortcuts import render
2 from django.http import HttpResponse
3
4 from cursos import controller
5
6 def index(request):
7     cursos = controller.ObtenerCursos()
8     return render(request, 'cursos/index.html',
9                   {
10                      "cursos": str(cursos)
11                   })
```

En nuestra página HTML solo hacemos referencia al parámetro que nos envió la vista

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <title>Cursos</title>
5 </head>
6 <body>
7     <h2>Mis cursos</h2>
8     <p>{{ cursos }}</p>
9 </body>
10 </html>
```





Formateando nuestro resultado

Podemos enviar directamente el resultado formateado desde nuestro BackEnd, como usamos MVC, el CONTROLADOR será el encargado de realizar esta tarea:

```
3 def ObtenerCursos():
4     Model = models.Model()
5     '''Obtenemos los cursos'''
6     result = Model.ObtenerCursosActivos()
7     '''Creamos una lista ul'''
8     result = "<ul>"
9     for curso in cursos:
10         for columna in curso:
11             result += "<li>" + str(columna) + "</li>"
12     result += "</ul>"
13     return result
```

Como estamos mandando código HTML, hay que hacer que nuestra vista realice el escape

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <title>Cursos</title>
5 </head>
6 <body>
7     <h2>Mis cursos</h2>
8     {% autoescape off %}{{ cursos }}{% endautoescape %}
9
10 </body>
11 </html>
```

Cursos x

← → ↻ 127.0.0.1:8000/ci

Mis cursos

- Python
- 5000.0000
- Isaac De la Rosa
- HTML
- 5500.0000
- Isaac De la Rosa
- JAVASCRIPT
- 3000.0000
- Isaac De la Rosa



Tag en HTML

Aunque es cómodo, es mala práctica enviar código HTML desde nuestro controlador, Django nos permite realizar estas operaciones directamente en nuestro HTML con los tags

Primero hay que hacer que el controlador nos devuelva un listado directo

```
1 from NOMBRE_DE_LA_APLICACION import models
2
3 def ObtenerCursos():
4     Model = models.Model()
5     '''Obtenemos los cursos'''
6     result = Model.ObtenerCursosActivos()
7     return result
```

En nuestra vista devolveremos directamente el listado:

```
1 from django.shortcuts import render
2 from django.http import HttpResponse
3
4 from cursos import controller
5
6 def index(request):
7     cursos = controller.ObtenerCursos()
8     return render(request, 'cursos/index.html',
9                   {
10                       "cursos": cursos
11                   })
```

Dentro de nuestro HTML, tendremos acceso a un listado de TAG HELPERS

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <title>Cursos</title>
5 </head>
6 <body>
7     <h2>Mis cursos</h2>
8     <ul>
9         {% for curso in cursos %}
10             {% for col in curso %}
11                 <li>{{col}}</li>
12             {% endfor %}
13         {% endfor %}
14     </ul>
15
16 </body>
17 </html>
```

