



Entrenamiento de Python-Django - 0

A partir de

Resumen: Hoy nos embarcaremos en un viaje para descubrir los conceptos básicos de la sintáctica y la semántica de Python.

Versión: 1

Contenido

| | | |
|--------------|--|----|
| I | Preámbulo | 2 |
| II | Reglas generales | 3 |
| III | Reglas específicas de hoy | 4 |
| IV | Ejercicio 00: mis primeras variables | 5 |
| V | Ejercicio 01 : Números | 6 |
| VI | Ejercicio 02: Mi primer diccionario | 7 |
| VII | Ejercicio 03: Búsqueda de claves | 9 |
| VIII | Ejercicio 04: Búsqueda por valor | 10 |
| Ejercicio IX | 05: Búsqueda por clave o valor | 11 |
| X | Ejercicio 06: Ordenamiento del diccionario | 12 |
| XI | Ejercicio 07: Tabla periódica de los elementos | 13 |
| XII | Presentación y evaluación por pares | 15 |

Capítulo I

Preámbulo

El zen de Python, por Tim Peters

Lo bello es mejor que lo feo.

Lo explícito es mejor que lo implícito.

Lo simple es mejor que lo complejo.

Lo complejo es mejor que lo complicado.

Lo plano es mejor que lo anidado.

Es mejor que lo disperso sea que lo denso.

La legibilidad cuenta.

Los casos especiales no son lo suficientemente especiales como para romper las reglas.

Aunque la practicidad supera a la pureza.

Los errores nunca deben pasar en silencio.

A menos que se silencie explícitamente.

Frente a la ambigüedad, rechaza la tentación de adivinar.

Debería haber una –y preferiblemente sólo una– manera obvia de hacerlo.

Aunque puede que ese camino no sea obvio al principio, a menos que seas holandés.

Ahora es mejor que nunca.

Aunque nunca es mejor que **ahora mismo**.

Si la implementación es difícil de explicar, es una mala idea.

Si la implementación es fácil de explicar, puede ser una buena idea.

Los espacios de nombres son una gran idea: ¡hagamos más de ellas!



importar esto

Capítulo II

Reglas generales

- Su proyecto debe realizarse en una máquina virtual.
- Su máquina virtual debe tener todo el software necesario para completar su proyecto. Estos software deben configurarse e instalarse.
- Puede elegir el sistema operativo que utilizará para su máquina virtual.
- Debe poder utilizar su máquina virtual desde una computadora del clúster.
- Debe utilizar una carpeta compartida entre su máquina virtual y su máquina host.
- Durante tus evaluaciones utilizarás esta carpeta para compartir con tu repositorio.

Sus funciones no deben cerrarse inesperadamente (fallo de segmentación, error de bus, doble liberación, etc.), salvo por comportamientos indefinidos. Si esto ocurre, su proyecto se considerará no funcional y recibirá un 0 en la evaluación.

Te animamos a crear programas de prueba para tu proyecto, aunque este trabajo no tenga que entregarse ni se califique. Esto te permitirá evaluar fácilmente tu trabajo y el de tus compañeros. Estas pruebas te resultarán especialmente útiles durante la defensa. De hecho, durante la defensa, puedes usar tus propias pruebas o las del compañero que estás evaluando.

Envía tu trabajo al repositorio Git asignado. Solo se calificará el trabajo en el repositorio Git. Si Deepthought se encarga de calificar tu trabajo, lo hará después de la evaluación por pares. Si se produce un error en alguna sección de tu trabajo durante la calificación de Deepthought, la evaluación se detendrá.

Capítulo III

Reglas específicas de hoy


- No hay código en el ámbito global. ¡Queremos funciones!
- Cada archivo entregado debe finalizar con una llamada de función en una condición idéntica a:

```
si __nombre__ == '__principal__':  
    your_function(cualquiera, parámetro, es, requerido)
```

- Puede establecer una gestión de errores en esta condición.
- No se autorizará ninguna importación, salvo las mencionadas explícitamente en el apartado 'Funciones autorizadas' de la descripción de cada ejercicio.
- No tendrás que gestionar las excepciones generadas por la función abierta.
- Tendrás que utilizar el intérprete de Python3.

Capítulo IV

Ejercicio 00: mis primeras variables

| | |
|---|--------------|
|  | Ejercicio 00 |
| Ejercicio 00: mis primeras variables | |
| Directorio de entrega: ex00/ | |
| Archivos a entregar: var.py | |
| Funciones permitidas: n/a | |

Crea un script llamado var.py donde definirás la función my_var. En esta función, declararás 9 variables de diferentes tipos y las imprimirás en la salida estándar. Reproducirás esta salida exactamente:


```
$> python3 var.py 42
tiene un tipo <class 'int'> 42 tiene un tipo
<class 'str'> quarante-deux tiene un tipo
<class 'str'> 42.0 tiene un tipo <class 'float'> True tiene un
tipo <class 'bool'> [42] tiene un tipo <class 'list'>
{42: 42} tiene un tipo <class 'dict'> (42,) tiene
un tipo <class 'tuple'> set() tiene un tipo
<class 'set'> $>
```

Por supuesto, está estrictamente prohibido escribir explícitamente los tipos de variables en las impresiones del código. No olvide llamar a su función al final del script, según lo requieran las instrucciones:

```
si __nombre__ == '__principal__':
    mi_var()
```

Capítulo V

Ejercicio 01: Números

| | |
|---|--------------|
|  | Ejercicio 01 |
| Ejercicio 01: Números | |
| Directorio de entrega: ex01/ | |
| Archivos a entregar: numbers.py | |
| Funciones permitidas: n/a | |


Para este ejercicio, eres libre de definir tantas funciones como desees y nombrarlas como quieras también.

El archivo tar d01.tar.gz en el apéndice de este tema contiene una subcarpeta ex01/ que contiene un archivo numbers.txt que contiene los números del 1 al 100 separados por una coma.

Diseña un script en Python llamado numbers.py cuya función es abrir un archivo numbers.txt, leer los números que contiene y mostrarlos en la salida estándar, uno por línea, sin coma.

Capítulo VI

Ejercicio 02: Mi primer diccionario

| | |
|---|--------------|
|  | Ejercicio 02 |
| Ejercicio 02: Mi primer diccionario | |
| Directorio de entrega: ex02/ | |
| Archivos a entregar: var_to_dict.py | |
| Funciones permitidas: NA | |

Una vez más, eres libre de definir tantas funciones como quieras y nombrarlas como quieras. Me gusta. No repetiremos esta instrucción, a menos que tenga que ser contradicha explícitamente.

Crea un script llamado var_to_dict.py en el que copiarás la siguiente lista de d parejas como esta en una de sus funciones:

```
d = [  
    ('Hendrix', '1942'),  
    ('Allman', '1946'),  
    ('Rey', '1925'),  
    ('Clapton', '1945'),  
    ('Johnson', '1911'),  
    ('Berry', '1926'),  
    ('Vaughan', '1954'),  
    ('Cooder', '1947'),  
    ('Página', '1944'),  
    ('Richards', '1943'),  
    ('Hammett', '1962'),  
    ('Cobain', '1967'),  
    ('García', '1942'),  
    ('Beck', '1944'),  
    ('Santana', '1947'),  
    ('Ramone', '1948'),  
    ('Blanco', '1975'),  
    ('Frusciante', '1970'),  
    ('Thompson', '1949'),  
    ('Burton', 1939) ,  
]
```


Su script debe convertir esta variable en un diccionario. El año será la clave y el nombre del músico, el valor. A continuación, debe mostrar este diccionario en la salida estándar con un formato claro:


```
1970: Frusciante  
1954: Vaughan  
1948: Ramone  
1944: Página Beck  
1911 : Johnson  
...
```



El orden final no tendrá que ser el mismo que en el ejemplo. Es un comportamiento habitual. ¿Sabes por qué?

Capítulo VII

Ejercicio 03: Búsqueda de claves

| | |
|---|--------------|
|  | Ejercicio 03 |
| Ejercicio 03: Búsqueda de claves | |
| Directorio de entrega: ex03/ | |
| Archivos a entregar: capital_city.py Funciones | |
| permitidas: import sys | |

Aquí tienes los diccionarios que debes copiar sin modificar en una de las funciones de tu script:

```
estados = {
    "Oregón": "OR",
    "Alabama": "AL",
    "Nueva Jersey": "NJ",
    "Colorado": "CO"
}


ciudades_capitales = {
    "O": "Salem",
    "AL": "Montgomery",
    "Nueva Jersey": "Trenton",
    "CO": "Denver"
}
```

Escriba un programa que tome un estado como argumento (p. ej., Oregón) y muestre su capital (p. ej., Salem) en la salida estándar. Si el argumento no genera ningún resultado, el script debe mostrar: Estado desconocido. Si no hay argumentos, o hay demasiados, el script no debe hacer nada y cerrar.

```
$> python3 capital_city.py Oregón Salem $> python3
capital_city.py Ile-De-France Estado desconocido $> python3 capital_city.py
$> python3 capital_city.py
Oregón Alabama $> python3 capital_city.py
Oregón Alabama Ile-De-France $>
```

Capítulo VIII

Ejercicio 04: Búsqueda por valor

| | |
|---|--------------|
|  | Ejercicio 04 |
| Ejercicio 04: Búsqueda por valor | |
| Directorio de entrega: ex04/ | |
| Archivos a entregar: state.py | |
| Funciones permitidas: import sys | |


Obtendrás los mismos diccionarios que en el ejercicio anterior. Tienes que copiarlos inalterado nuevamente en una de las funciones de tu script.

Cree un programa que tome la capital como argumento y muestre el estado correspondiente esta vez. El resto del comportamiento del programa debe ser el mismo que en el ejercicio anterior.

```
$> python3 state.py Salem Oregon $>
python3
state.py París Capital desconocida $>
python3 state.py $>
```

Capítulo IX

Ejercicio 05: Búsqueda por clave o valor

| | |
|---|--------------|
|  | Ejercicio 05 |
| Ejercicio 05: Búsqueda por clave o valor | |
| Directorio de entrega: ex05/ | |
| Archivos a entregar: all_in.py | |
| Funciones permitidas: import sys | |


Comenzando con los mismos diccionarios, debes copiarlos nuevamente sin modificarlos en una de tus funciones de script y escribir un programa que se comporte de la siguiente manera:

- El programa debe tomar como argumento una cadena que contenga tantas expresiones como Buscamos, separados por una coma.
- Para cada expresión en esta cadena, el programa debe detectar si es una capital, un estado o ninguno de ellos.
- El programa no debe distinguir entre mayúsculas y minúsculas. No debe contener varios espacios en blanco. consideración tampoco.
- Si no hay ningún parámetro o hay demasiados parámetros, el programa no muestra cualquier cosa.
- Cuando hay dos comas sucesivas en la cadena, el programa no muestra cualquier cosa.
- El programa debe mostrar los resultados separados por un retorno de carro y utilizar estrictamente el siguiente formato:

```
$> python3 all_in.py "Nueva Jersey, Trenton, Nueva Jersey, Trenton, toto, Trenton es la capital de Nueva Jersey, Salem"
Nueva Jersey no es ni una ciudad capital ni un estado
Trenton es la capital de Nueva Jersey
toto no es ni una ciudad capital ni un estado
Salem es la capital de Oregón $>
```

Capítulo X

Ejercicio 06: Ordenamiento del diccionario

| | |
|---|--------------|
|  | Ejercicio 06 |
| Ejercicio 06: Ordenamiento del diccionario | |
| Directorio de entrega: ex06/ | |
| Archivos a entregar: my_sort.py | |
| Funciones permitidas: NA | |


Integra este diccionario en cualquiera de tus funciones como:

```
d = {
    'Hendrix': '1942',
    'Allman': '1946',
    'Rey' : '1925',
    'Clapton': '1945',
    'Johnson': '1911',
    'Baya' : '1926',
    'Vaughan': '1954',
    'Cooder': '1947',
    'Página' '1944', :
    'Richards': '1943',
    'Hammett': '1962',
    'Cobain': '1967',
    'Garcia' : '1942',
    'Arroyo' : '1944',
    'Santana' : '1947',
    'Ramone' : '1948',
    'Blanco' : '1975',
    'Frusciante': '1970',
    'Thompson': '1949',
    'Burton': '1939',
}
```

Escriba un programa que muestre los nombres de los músicos ordenados por año en orden ascendente. Ordenar, luego en orden alfabético para años similares. Uno por línea, sin mostrar el año.

Capítulo XI

Ejercicio 07: Tabla periódica de los elementos

| | |
|---|--------------|
|  | Ejercicio 07 |
| Ejercicio 07: Tabla periódica de los elementos | |
| Directorio de entrega: ex07/ | |
| Archivos a entregar: periodic_table.py Funciones | |
| permitidas: import sys | |

El archivo tar d01.tar.gz en el apéndice de este tema contiene la subcarpeta ex07/ en la que encontrará el archivo periodic_table.txt, que describe una tabla periódica de los elementos en un formato hecho para programadores.

Cree un programa que utilice el archivo para escribir una página HTML que represente la información periódica. Tabla de los elementos en un formato adecuado.

- Cada elemento debe estar en un "cuadro" de una tabla HTML.
- El nombre de un elemento debe estar en una etiqueta de título de nivel 4.

Los atributos de un elemento deben presentarse como una lista. Las listas deben indicar al menos los números atómicos, el símbolo y la masa atómica.

Debes al menos respetar el diseño de la Tabla de Mendeleiev tal como aparece en Google. Debe haber casillas vacías donde debería haberlas, así como retornos de carro donde debería haberlos.

Su programa debe crear el archivo de resultados tabla_periodica.html. Por supuesto, este archivo HTML debe ser legible en cualquier navegador y debe ser compatible con el W3C.

Tienes la libertad de diseñar tu programa como quieras. No dudes en fragmentar tu código en funcionalidades específicas que puedas reutilizar. Puedes personalizar las etiquetas con un estilo CSS "en línea" para que tu entrega sea más atractiva (piensa en los colores de los bordes de la tabla). Incluso puedes...

Genere el archivo `periodic_table.css` si lo prefiere.

A continuación se muestra un extracto de un ejemplo de salida que le dará una idea:

```
[...]  
<tabla>  
  <tr>  
    <td style="borde: 1px negro sólido; relleno: 10px">  
      <h4>Hidrógeno</h4>  
      <ul>  
        <li>No 42</li> <li>H</li>  
        <li>1,00794</li>  
        <li>1 electrón</li>  
      </ul>  
    </td> [...]
```

Capítulo XII

Presentación y evaluación por pares

Entrega tu tarea en tu repositorio de Git como de costumbre. Solo se evaluará el trabajo dentro de tus repositorios durante la defensa. No dudes en verificar los nombres de tus carpetas y archivos para asegurarte de que sean correctos.



El proceso de evaluación se realizará en el computador del evaluado.
grupo.