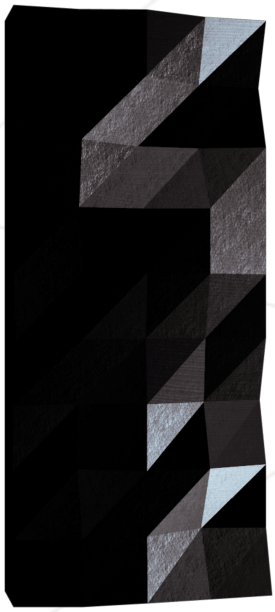

Entrenamiento Piscine Python para datascience - 0

A partir de

Resumen: Hoy aprenderás los conceptos básicos del lenguaje de programación Python.

Versión: 1.1

Contenido

Capítulo I

Reglas generales

- Debes renderizar tus módulos desde una computadora en el clúster usando una máquina virtual:
    - Puede elegir el sistema operativo que utilizará para su máquina virtual
    - Su máquina virtual debe contar con todo el software necesario para realizar su proyecto. Este software debe estar configurado e instalado.
- O puedes utilizar directamente la computadora en caso de que tengas las herramientas disponibles.
    - Asegúrate de tener el espacio en tu sesión para instalar lo que necesitas para todos los módulos (usa goinfre si tu campus lo tiene)
    - Debes tener todo instalado antes de las evaluaciones.
- Sus funciones no deberían cerrarse inesperadamente (error de segmentación, error de bus, doble liberación, etc.) salvo por comportamientos indefinidos. Si esto sucede, su proyecto se considerará no funcional y recibirá una0Durante la evaluación.
- Le animamos a crear programas de prueba para su proyecto, incluso si este trabajo no es suficiente. **No será necesario enviarlo y no será calificado.**. Te dará la oportunidad de evaluar fácilmente tu trabajo y el de tus compañeros. Estas pruebas te resultarán especialmente útiles durante tu defensa. De hecho, durante la defensa, eres libre de utilizar tus propias pruebas y/o las pruebas de los compañeros a los que estás evaluando.
- Envía tu trabajo al repositorio git que se te haya asignado. Solo se calificará el trabajo que se encuentre en el repositorio git. Si se le asigna a Deepthought la tarea de calificar tu trabajo, se hará después de las evaluaciones de tus pares. Si ocurre un error en alguna sección de tu trabajo durante la calificación de Deepthought, la evaluación se detendrá.
- Debes utilizar la versión Python 3.10
- Puedes utilizar cualquier función incorporada si no está prohibida en el ejercicio.
- Las importaciones de bibliotecas deben ser explícitas, por ejemplo, debe "importar numpy como np". No se permite importar "from pandas import *" y obtendrá 0 en el ejercicio.
- No hay ninguna variable global.
- ¡Por Odín, por Thor! ¡Usa tu cerebro!

Capítulo II

Ejercicio 00

	Ejercicio 00
Ejercicio 00: Primer script de Python	
Directorio de entregas: *ex*00/	
Archivos a entregar:Hola.py	
Funciones permitidas:Ninguno	

Debes modificar la cadena de cada objeto de datos para mostrar los siguientes saludos: "Hola mundo", "Hola «país de tu campus»", "Hola «ciudad de tu campus»", "Hola «nombre de tu campus»".

```
lista de pies=["Hola","¡Joder!"]
tupla_ft=("Hola","¡Toto!") conjunto
de pies      ={"Hola","¡tutú!"} ={"Hola"
ft_dict      :"¡Titi!"}

# tu código aquí

imprimir(ft_list)
imprimir(ft_tuple)
imprimir(ft_set)
imprimir(ft_dict)
```

Resultado esperado:

```
$>python Hello.py | cat -e
['Hola', '¡Mundo!']$
('¡Hola, Francia!')$
{'¡Hola, París!'}$
{'Hola': '42Paris!'}$
$>
```

Capítulo III

Ejercicio 01

	Ejercicio 01
Ejercicio 01: Primer uso del paquete	
Directorio de entregas:*ex*01/	
Archivos a entregar:formato_ft_time.py	
Funciones permitidas:hora, fecha y hora o cualquier otra biblioteca que permita recibir la fecha	

Escribe un script que formatee las fechas de esta manera, por supuesto tu fecha no será la mía como en el ejemplo pero debe tener el mismo formato.

Resultado esperado:

```
>python format_ft_time.py | cat -e
Segundos desde el 1 de enero de 1970: 1.666.355.857,3622 o 1,67e+09 en notación científica$ 21 de octubre
de 2022$
$>
```

Capítulo IV

Ejercicio 02

	Ejercicio 02
Ejercicio 02: Primera función python	
Directorio de entregas:*ex*02/	
Archivos a entregar:buscar_tipo_ft.py	
Funciones permitidas:Ninguno	

Escriba una función que imprima los tipos de objetos y devuelva 42.

Así es como debería hacerse el prototipo:

```
def todo_es_obj(objeto:cualquier)->entero:
        # tu código aquí
```

Su tester.py:

```
desde find_ft_type importa todo_es_obj

lista de pies=["Hola","¡Joder!"]
tupla_ft=("Hola","¡Toto!") conjunto
de pies      ={"Hola","¡tutú!"} ={"Hola"
ft_dict      :"¡Titi!"}

todo_es_obj(ft_lista)
todo_es_obj(ft_tupla)
todo_es_obj(ft_conjunto)
todo_es_obj(ft_diccionario)
todo_es_obj("Brian")
todo_es_obj("Toto")
imprimir(todo_es_obj(10))
```

Resultado esperado:

```
$>python tester.py | cat -e Lista:
<clase 'lista'>$ Tupla: <clase
'tupla'>$ Conjunto: <clase
'conjunto'>$
Diccionario: <clase 'dict'>$
Brian está en la cocina: <class 'str'>$ Toto está
en la cocina: <class 'str'>$ Tipo no encontrado$

42$
$>
```

Ejecutar su función por sí solo no hace nada.

Resultado esperado:

```
$>python find_ft_type.py | cat -e $>
```

Capítulo V

Ejercicio 03

	Ejercicio 03
Ejercicio 03: NULL no encontrado	
Directorio de entregas:*ex*03*/*	
Archivos a entregar:NULL_no_encontrado.py	
Funciones permitidas:Ninguno	

Escriba una función que imprima el tipo de objeto de todos los tipos "Null".
Devuelva 0 si todo va bien y 1 en caso de error.
Su función debe imprimir todos los tipos de NULL.

Así es como debería hacerse el prototipo:

```
def NULL_no_encontrado(objeto:cualquier)->entero:
  # tu código aquí
```

Su tester.py:

```
de NULL_no_encontrado importar NULL_no_encontrado

Nada=Ninguno
Ajo      =flotar("Yaya") =
Cero     0
Vacío    ="
Falso    =FALSO

NULL_not_found(Nada)
NULL_not_found(Ajo)
NULL_no_encontrado(cero)
NULL_not_found(Vacío)
NULL_not_found(Falso)
imprimir(NULL_no_encontrado("Brian"))
```

Vacío = ”

Resultado esperado:

```
$>python tester.py | cat -e Nada:
Ninguno <clase 'NoneType'>$ Queso:
nan <clase 'float'>$ Cero: 0 <clase 'int'>$

Vacío: <class 'str'>$
Falso: Falso <class 'bool'>$ Tipo
no encontrado$
1$
$>
```

Ejecutar su función por sí solo no hace nada.

Resultado esperado:

```
$>python NULL_no_encontrado.py | cat -e $>

```

Capítulo VI

Ejercicio 04

	Ejercicio 04
Ejercicio 04: El par y el impar	
Directorio de entregas:*ex*04*/*	
Archivos a entregar:¿Qué es? py	
Funciones permitidas:sys o cualquier otra biblioteca que permita recibir los argumentos	

Crea un script que tome un número como argumento, verifique si es par o impar e imprima el resultado.

Si se proporciona más de un argumento o si el argumento no es un entero, imprima un **Error de afirmación**.

Resultado esperado:

```
$> python whatis.py 14
Estoy parejo.
$>
$> python whatis.py -5 Soy
extraño.
$>
> python ¿qué es? python
$>
$> python whatis.py 0
Estoy parejo.
$>
$> python whatis.py Hola! AssertionError: el
argumento no es un entero $>

> python qué es.py 13 5
AssertionError: se proporcionó más de un argumento $>
```

Capítulo VII

A partir de ahora deberás seguir estas reglas adicionales

- No hay código en el ámbito global. ¡Utilice funciones!
- Cada programa debe tener su main y no ser un simple script:

```
definición principal():
    # sus pruebas y su manejo de errores

si __nombre__ == "__principal__":
    principal()
```

- Cualquier excepción no detectada invalidará los ejercicios, incluso en el caso de un error que se le solicitó probar.
- Todas tus funciones deben tener una documentación (__doc__)
- Su código debe estar en la norma
    - pip instala flake8
    - alias norminette=flake8

Capítulo VIII

Ejercicio 05

	Ejercicio 05
Ejercicio 05: Primer programa independiente en Python	
Directorio de entregas:*ex*05/	
Archivos a entregar:edificio.py	
Funciones permitidas:sys o cualquier otra biblioteca que permita recibir los argumentos	

Esta vez tienes que hacer un programa autónomo real, con un principal, que toma un único argumento de cadena y muestra las sumas de sus caracteres en mayúsculas, minúsculas, caracteres de puntuación, dígitos y espacios.

- Si no se proporciona Ninguno o nada, se le solicita al usuario que proporcione una cadena.
- Si se proporciona más de un argumento al programa, imprima un**Error de afirmación**.

Resultados esperados:

```
$>python building.py "Python 3.0, lanzado en 2008, fue una revisión importante que no es completamente retroactiva.
        compatible con versiones anteriores. Python 2 se discontinuó con la versión 2.7.18 en 2020. El texto
contiene 171 caracteres:
2 letras mayúsculas
121 letras minúsculas
8 signos de puntuación
25 espacios
15 dígitos
$>
```

Resultados esperados: (el retorno de carro cuenta como un espacio, si no desea devolver uno, use Ctrl + D)

```
>construcción de python.py
¿Cuál es el texto a contar? ¡Hola
mundo!
El texto contiene 13 caracteres: 2
letras mayúsculas
8 letras minúsculas
1 signos de puntuación
2 espacios
0 dígitos
$>
```

¡Por Odín, por Thor! ¡Usa tu cerebro! No reinventes la rueda, usa las funciones del lenguaje.

Capítulo IX

Ejercicio 06

	Ejercicio 06
Ejercicio 06:	
Directorio de entregas: *ex*06/	
Archivos a entregar:ft_filter.py, cadena de filtro.py	
Funciones permitidas:sys o cualquier otra biblioteca que permita recibir los argumentos	

Parte 1: Función de filtro de recodificación

Recodifica tu propio ft_filter, debería comportarse como la función incorporada original (debería devolver lo mismo que "print(filter.__doc__)"), deberías usar**listas por comprensión** para recodificar su ft_filter.

Por supuesto, está prohibido utilizar el filtro original incorporado.

Puedes validar el módulo desde aquí, pero te animamos a que continúes ya que hay cosas que necesitarás saber para los siguientes proyectos.

Parte 2: El programa

Cree un programa que acepte dos argumentos: una cadena (S) y un entero (N). El programa debe generar una lista de palabras de**S**que tienen una longitud mayor que**norte**.
• Las palabras están separadas entre sí por caracteres de espacio.
• Las cadenas no contienen ningún carácter especial (puntuación o invisible).
• El programa debe contener al menos uno**comprensión de listas**expresión y uno **lambda**.
• Si el número de argumentos es diferente de 2, o si el tipo de algún argumento es incorrecto, el programa imprime un**Error de afirmación**.

Resultados esperados:

```
$> python filterstring.py 'Hola mundo' 4 ['Hola',
'Mundo']
$>
```

```
$> python filterstring.py 'Hola mundo' 99 []

$>
```

```
$> python filterstring.py 3 'Hola mundo'
AssertionError: los argumentos son incorrectos $>
```

```
$> python filterstring.py AssertionError: los
argumentos son incorrectos $>
```

Capítulo X

Ejercicio 07

	Ejercicio 07
Ejercicio 07: Diccionarios SoS	
Directorio de entregas:*ex*07*/*	
Archivos a entregar:sos.py	
Funciones permitidas:sys o cualquier otra biblioteca que permita recibir los argumentos	

Cree un programa que tome una cadena como argumento y la codifique enCódigo Morse.

- El programa admite espacios y caracteres alfanuméricos.
- Un carácter alfanumérico se representa mediante puntos y guiones.
- Los caracteres morse completos están separados por un solo espacio.
- Un carácter de espacio se representa mediante una barra /

Debes utilizar un**diccionario**para almacenar su código morse.

```
MORSE ANIDADO={      " ":"/ ",
                     "A":".- ",
                     . . .
```

Si el número de argumentos es diferente de 1, o si el tipo de algún argumento es incorrecto, el programa imprime un**Error de afirmación**.

```
$> python sos.py "sos" | cat -e . . . - -
- . . . $
$> python sos.py 'h$llo' AssertionError: los
argumentos son incorrectos $>
```

Capítulo XI

Ejercicio 08

	Ejercicio 08
Ejercicio 08: Cargando...	
Directorio de entregas:*ex*08/	
Archivos a entregar:Cargando.py	
Funciones permitidas:Ninguno	

Así que vamos a crear una función llamadaarchivo .ft_tqdm.

La función debe copiar la función tqdm con laproduciroperador.

Así es como debería hacerse el prototipo:

```
definición ft_tqdm(lst:rango)->Ninguno:
  # tu código aquí
```

Tu tester.py: (comparas tu versión con la original)

```
desde tiempo importar dormir desde
tqdm importar tqdm desde Cargando
importar ft_tqdm

paraelemento en ft_tqdm(rango(333)):
        dormir(0,005)
imprimir()
paraelemento en tqdm(rango(333)):
        dormir(0,005)
imprimir()
```

Resultado esperado: (debe tener una función lo más cercana posible a la versión original)

```
    > probador de python.py
100%|[============================================== ================>]| 333/333
100%|                                                                  | 333/333 [00:01<00:00, 191,61 it/s]
```

Capítulo XII

Ejercicio 09

	Ejercicio 09
Ejercicio 09: Mi primer creación de paquete	
Directorio de entregas: *ex*09/	
Archivos a entregar: *.py, *.txt, *.toml, README.md, LICENCIA	
Funciones permitidas:PyPI o cualquier biblioteca para la creación de paquetes	

Crea tu primer paquete en python de la forma que desees, este aparecerá en la lista de paquetes instalados cuando escribas el comando "pip list" y mostrará sus características cuando escribas "pip show -v ft_package"

```
$>pip show -v ft_package
Nombre: ft_package
Versión: 0.0.1
Resumen: Un paquete de prueba de muestra
Página de inicio: https://github.com/eagle/ft_package
Autor: eagle
Autor-email: eagle@42.fr
Licencia: MIT
Ubicación: /home/eagle/...
Requiere:
Requerido por:
Versión de metadatos: 2.1
Instalador: pip
Clasificadores:
Puntos de entrada:
$>
```

El paquete se instalará a través de pip usando uno de los siguientes comandos (ambos deberían funcionar):

• instalación de pip ./dist/ft_package-0.0.1.tar.gz
• instalación de pip ./dist/ft_package-0.0.1-py3-none-any.whl

Su paquete debe poder llamarse desde un script como este:

```
desde ft_package importar count_in_list

imprimir(contar_en_lista(["Toto","Tía","Toto"],"Toto")) # producción: 2
imprimir(contar_en_lista(["Toto","Tía","Toto"],"tutú")) # producción: 0
```

Capítulo XIII

Presentación y evaluación por pares

Entregue su tarea en suGitRepositorio como de costumbre. Solo el trabajo dentro de su repositorio será evaluado durante la defensa. No dude en volver a verificar los nombres de sus carpetas y archivos para asegurarse de que sean correctos.

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