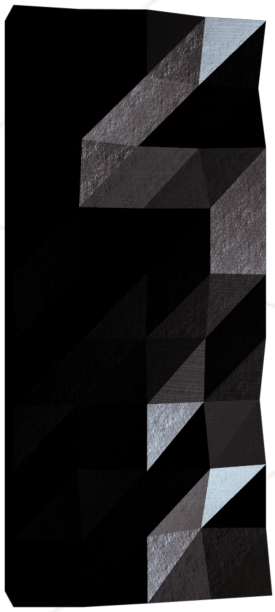

Entrenamiento Piscine Python para datascience - 3

Programación Orientada a Objetos

Resumen: Hoy conoceréis las clases y el patrimonio.

Versión: 1.00

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
- 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

Instrucciones específicas del día

Una queja común de los científicos de datos es que escriben código basura (por cierto, solo con fines educativos puedes encontrar muchos ejemplos de código basura de Python).aquí, proporcionado estrictamente con fines educativos). ¿Por qué? Porque el científico de datos promedio utiliza muchas técnicas ineficientes y variables codificadas de forma rígida y descuida la programación orientada a objetos. No sea como ellos.

- 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.
- Puedes utilizar cualquier función incorporada si no está prohibida en el ejercicio.
- Todas sus funciones, clases y métodos deben tener una documentación (__doc__)
- Su código debe estar en la norma
    - pip instala flake8
    - alias norminette=flake8

Capítulo III

Ejercicio 00

	Ejercicio 00
Ejercicio 00: GOT T1E9	
Directorio de entregas:*ex*00/	
Archivos a entregar:S1E9.py	
Funciones permitidas:Ninguno	

Cree una clase abstracta "carácter" que pueda tomar un nombre como primer parámetro, is_alive como segundo parámetro no obligatorio establecido en Verdadero de manera predeterminada y pueda cambiar el estado de salud del personaje con un método que pase is_alive de Verdadero a Falso.

Y una clase "stark" que hereda de Character El prototipo de Clase es:

```
de abc importa ABC, método abstracto

Clase Carácter(ABC):
    """Su cadena de documentación para la clase"""
    @metodoabstracto
        # tu código aquí

Clase Stark(Personaje):
    """Su cadena de documentación para la clase"""
        # tu código aquí
```

Su tester.py:

```
Del personaje de importación S1E9, Stark

Ned=Rígido("Ned")
imprimir(Ned.__dict__)
imprimir(Ned.está_vivo)
Ned.muere()
imprimir(Ned.está_vivo)
imprimir(Ned.__doc__)
imprimir(Ned.__init__.__doc__)
imprimir(Ned.die.__doc__)
imprimir("---")
Lyanna=Rígido("Liana", Falso)
imprimir(Lyanna.__dict__)
```

Resultado esperado: (las cadenas de documentación pueden ser diferentes)

```
> probador de python.py
{'first_name': 'Ned', 'is_alive': Verdadero}
Verdadero
FALSO
Su cadena de documentación para la clase Su cadena
de documentación para el constructor Su cadena de
documentación para el método
- - -
{'first_name': 'Lyanna', 'is_alive': Falso}
$>
```

Asegúrese de haber utilizado una clase abstracta, el código a continuación debería generar un error.

```
de S1E9 importar personaje

hodor=Personaje("hodor")
```

```
TypeError: No se puede crear una instancia de la clase abstracta Character con un método abstracto
$>
```

Capítulo IV

Ejercicio 01

	Ejercicio 01
Ejercicio 01: GOT T1E7	
Directorio de entregas:*ex*01/	
Archivos a entregar:Archivos de ejercicios anteriores + S1E7.py	
Funciones permitidas:Ninguno	

Crea dos familias que hereden de la clase Character, que podamos instanciar sin pasar por la clase Character. Encuentra una solución para que "__str__" y "__repr__" devuelvan cadenas y no objetos. Escribe un método Class para crear caracteres en una cadena.

El prototipo de clase es:

```
de S1E9 importar personaje

Clase Baratheon(Personaje):
  # tu código aquí

Clase Lannister(Personaje):
  # tu código aquí

            # decorador
      def create_lannister(tu código aquí):
        # tu código aquí
```

Su tester.py:

```
De la temporada 1, episodio 7, importación Baratheon, Lannister

Roberto=Baratheon("Roberto")
imprimir(Robert.__dict__)
imprimir(Robert.__str__)
imprimir(Robert.__repr__)
imprimir(Robert.está_vivo)
Roberto.muere()
imprimir(Robert.está_vivo)
imprimir(Robert.__doc__)
imprimir("---")
Cersei=Lannister("Cersei")
imprimir(Cersei.__dict__)
imprimir(Cersei.__str__)
imprimir(Cersei.está_viva)
imprimir("---")
Jaine=Lannister.create_lannister("Jaine", Verdadero)
imprimir(f"Nombre: {Jaine.first_name, tipo(Jaine).__name__}, Viva: {Jaine.is_alive}")
```

Resultado esperado: (las cadenas de documentación pueden ser diferentes)

```
> probador de python.py
{'first_name': 'Robert', 'is_alive': True, 'family_name': 'Baratheon', 'eyes': 'brown', 'hairs': 'dark'} <método enlazado
Baratheon.__str__ de Vector: ('Baratheon', 'brown', 'dark')>
<método enlazado Baratheon.__repr__ de Vector: ('Baratheon', 'marrón', 'oscuro')>
Verdadero
FALSO
Representando a la familia Baratheon.
- - -
{'first_name': 'Cersei', 'is_alive': True, 'family_name': 'Lannister', 'eyes': 'blue', 'hairs': 'light'} <método enlazado Lannister.__str__
de Vector: ('Lannister', 'blue', 'light')>
Verdadero
- - -
Nombre: ('Jaine', 'Lannister'), Vivo: Verdadero $>
```

Capítulo V

Ejercicio 02

	Ejercicio 02
Ejercicio 02: ¡Ahora es raro!	
Directorio de entregas:*ex*02/	
Archivos a entregar:Archivos de ejercicios anteriores + DiamondTrap.py	
Funciones permitidas:Ninguno	

En este ejercicio, crearás un monstruo: Joffrey Baratheon. ¡Es muy arriesgado!
Hay algo incoherente con este nuevo “falso” rey.
Debes usar las Propiedades para cambiar las características físicas de nuestro nuevo rey. El prototipo de Clase es:

```
De la temporada 1, episodio 7, importación Baratheon, Lannister

Clase Rey (Baratheon, Lannister):
  # tu código aquí
```

Su tester.py:

```
de DiamondTrap import King

Joffrey=Rey("Joffrey")
imprimir(Joffrey.__dict__)
Joffrey.establecer_ojos("azul")
Joffrey.set_hairs("luz")
imprimir(Joffrey.obtener_ojos())
imprimir(Joffrey.obtener_cabellos())
imprimir(Joffrey.__dict__)
```

Resultado esperado: (las cadenas de documentación pueden ser diferentes)

```
> probador de python.py
{'first_name': 'Joffrey', 'is_alive': Verdadero, 'family_name': 'Baratheon', 'eyes': 'brown', 'hair': 'dark'} azul

luz
{'first_name': 'Joffrey', 'is_alive': Verdadero, 'family_name': 'Baratheon', 'eyes': 'azul', 'hairs': 'claro'} $>
```

Desde Python 2.3, el lenguaje utiliza la linealización C3 para contrarrestar el problema de la herencia en diamand.

Capítulo VI

Ejercicio 03

	Ejercicio 03
Ejercicio 03: Calcular mi vector	
Directorio de entregas:*ex*03*/*	
Archivos a entregar:calculadora_ft.py	
Funciones permitidas:Ninguno	

Escriba una clase de calculadora que pueda realizar cálculos (suma, multiplicación, resta, división) de vectores con un escalar.
El prototipo de clase es:

```
calculadora de clases:
  # tu código aquí

            def __add__(yo, objeto)->Ninguno:
                        # tu código aquí
            def __mul__(yo, objeto)->Ninguno:
                        # tu código aquí
            def __sub__(yo, objeto)->Ninguno:
                        # tu código aquí
            def __truediv__(yo, objeto)->Ninguno:
                        # tu código aquí
```

Su tester.py:

```
desde ft_calculator importar calculadora

versión 1=calculadora([0.0,1.0,2.0,3.0,4.0,5.0]) versión 1+
5
Imprimir("---")
versión 2=calculadora([0.0,1.0,2.0,3.0,4.0,5.0]) versión 2*
5
Imprimir("---")
versión 3=calculadora([10.0,15.0,20.0])
versión 3- 5
versión 3/ 5
```

Resultado esperado: (las cadenas de documentación pueden ser diferentes)

```
> probador de python.py
[5,0, 6,0, 7,0, 8,0, 9,0, 10,0]
- - -
[0,0, 5,0, 10,0, 15,0, 20,0, 25,0]
- - -
[5.0, 10.0, 15.0]
[1.0, 2.0, 3.0]
$>
```

No es necesario realizar ningún tipo de gestión de errores, excepto la división por 0.

Capítulo VII

Ejercicio 04

	Ejercicio 04
Ejercicio 04: Calcular mi producto escalar	
Directorio de entregas:*ex*04*/*	
Archivos a entregar:calculadora_ft.py	
Funciones permitidas:Ninguno	

Escriba una clase de calculadora que pueda realizar cálculos (producto escalar, suma, resta) de 2 vectores.

El vector siempre tendrá tamaños idénticos, sin manejo de errores.

Depende de usted encontrar un decorador que pueda ayudarlo a utilizar los métodos de la clase calculadora sin instanciar esta clase.

El prototipo de clase es:

```
calculadora de clases:
  # tu código aquí

        # decorador
        def producto_punto(V1:lista[flotar], V2:lista[flotar])->Ninguno:
                # tu código aquí

        # decorador
        definición agregar_vec(V1:lista[flotar], V2:lista[flotar])->Ninguno:
                # tu código aquí

        # decorador
        def sub_vec(V1:lista[flotar], V2:lista[flotar])->Ninguno:
                # tu código aquí
```

Su tester.py:

```
desde ft_calculator importar calculadora

a=[5,10,2] b=[2,4,3]
calculadora.producto_punto(a,b)
calculadora.suma_vec(a,b)
calculadora.sub_vec(a,b)
```

Resultado esperado: (las cadenas de documentación pueden ser diferentes)

```
> probador de python.py
El producto escalar es: 56
Agregar vector es: [7.0, 14.0, 5.0]
Subvector es: [3.0, 6.0, -1.0] $>
```

No es necesario realizar ningún tipo de gestión de errores.

Si deseas profundizar en los cálculos vectoriales o matriciales ve al proyecto matricial después de esta "piscina".

Capítulo VIII

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.