Clase8_1_Iteradores

May 4, 2022

0.0.1 Seminario de Lenguajes - Python

0.1 Cursada 2022

0.1.1 Iteradores - Definición de nuevas excepciones

1 ¿Qué observan en el siguiente código?

[ ]:
```
cadena = "Seminario de Python"
for caracter in cadena:
    print(caracter)

lista = ['esto', 'es', 'una', 'lista']
for palabra in lista:
    print(palabra)

superheroes = { 'Ironman' : 'Marvel', 'Batman' : 'DC'}
for clave, valor in superheroes.items():
    print("{} : {}". format(clave, valor))

for linea in open("imagine.txt"):
    print(linea)
```

2 Todas son secuencias iterables

2.1 ¿Qué significa?

- Todas pueden ser recorridas por la estructura: **for** var **in** secuencia.
- Todas implementan un método especial denominado **___iter___**.
  - ___iter___ devuelve un iterador capaz de recorrer la secuencia.

Un **iterador** es un objeto que permite recorrer uno a uno los elementos de una estructura de datos para poder operar con ellos.

3 Iteradores

- Un iterador tiene que implementar un método **___next___** que debe devolver los elementos, de a uno por vez, comenzando por el primero.

- Y al llegar al final de la estructura, debe levantar una excepción de tipo **StopIteration**.

4 Los siguientes códigos son equivalentes:

[ ]:
```
lista = ['uno', 'dos', 'tres']
for palabra in lista:
    print(palabra)
```

[ ]:
```
iterador = iter(lista)
while True:
    try:
        palabra = next(iterador) # o iterador.__next__()
    except StopIteration:
        break
    print(palabra)
```

- La función **iter** retorna un objeto iterador.

5 Veamos este ejemplo:

[ ]:
```
class CadenaInvertida:
    def __init__(self, cadena):
        self._cadena = cadena
        self._posicion = len(cadena)

    def __iter__(self):
        return(self)

    def __next__(self):
        if self._posicion == 0:
            raise(StopIteration)
        self._posicion = self._posicion - 1
        return(self._cadena[self._posicion])

cadena_invertida = CadenaInvertida('Seminario de Python')

for caracter in cadena_invertida:
     print(caracter, end=' ')
```

- ¿Qué creen que imprime?

6 Podemos definir nuestras excepciones

[ ]:
```python
class ExcepcionRara(Exception):
    """ Esta excepción se producirá cuando ....."""

    def __init__(self, valor):
        self.data = valor

    def __str__(self):
        return f"Info: {self.data}"
```

- Las excepciones definidas por el usuario deberán derivar de la clase Exception, directa o indirectamente.

7 ¿Cómo levantamos esta excepción?

- Como levantamos cualquier otra excepción: **raise**

[ ]:
```python
try:
    raise ExcepcionRara("Hola mundo")

except ExcepcionRara as e:
    print(f"Ups! Se produjo la excepción rara!! {e}")
```

8 Algunas convenciones

- Según la PEP 8, el nombre de de la clase debería tener el sufijo "Error", **si la excepción representa un error**.
- Veamos este ejemplo de la documentación oficial:

[ ]:
```python
class Error(Exception):
    """Base class for exceptions in this module."""
    pass

class InputError(Error):
    """Exception raised for errors in the input.
    Attributes:
        expression -- input expression in which the error occurred
        message -- explanation of the error
    """

    def __init__(self, expression, message):
        self.expression = expression
        self.message = message
```

[ ]:
```
try:
    raise InputError("xxx","hola")
except InputError as e:
    print(e)
```

9 Al crear un módulo que puede producir distintos errores, se puede crear una clase base para las excepciones definidas en ese módulo y extenderla para crear clases derivadas correspondientes.