Python 3 para impacientes

"Simple es mejor que complejo" (Tim Peters)

martes, 31 de marzo de 2020

Enumeraciones

Una **enumeración** es una agrupación de constantes que guardan alguna relación y que comparten un mismo espacio de nombres. Las enumeraciones dan consistencia a los programas porque evitan que las constantes sean objetos aislados a los que cueste identificar con el paso del tiempo. También, proporcionan una metodología para abordar con un mismo criterio de estructuración del código implementaciones que se dan con frecuencia.

Las enumeraciones se pueden utilizar para agrupar los diferentes estados de un proceso; las distintas respuestas que pueden darse en los condicionantes que determinan el flujo de un programa; las constantes matemáticas o físicas necesarias para realizar determinados cálculos en un desarrollo y los valores o características de objetos que son del mismo tipo, entre otros.

Creación de una enumeración

Para declarar una enumeración utilizaremos la clase **Enum** del módulo **enum**, un viejo conocido en Python como módulo independiente que pasó a formar parte de la librería estándar a partir de Python 3.4 y que ha sufrido alguna ampliación de sus funcionalidades en Python 3.6. A continuación, algunos ejemplos de declaración de enumeraciones:

```
from enum import Enum


class Respuesta(Enum):
    SI = 1
    NO = 2


class Estado(Enum):
    DETENIDO = 0
    ENESPERA = 0
    INICIADO = 1


class Andalucia(Enum):
    AL = (715993, 8775)
    CA = (1240020, 7436)
    CO = (782516, 13771)
    GR = (914428, 12647)
    HU = (521428, 10128)
    JA = (633120, 13496)
    MA = (1660693, 7308)
    SE = (1941804, 14036)


Semana = Enum(
    value='Semana',
    names=('LU MA MI JU VI SA DO'),
```

Buscar

Buscar

Python para impacientes

Python
IPython
EasyGUI
Tkinter
JupyterLab
Numpy

Anexos

Guía urgente de MySQL
Guía rápida de SQLite3

Entradas + populares

Dar color a las salidas en la consola
En Python para dar color a las salidas en la consola (o en la terminal de texto) existen varias posibilidades. Hay un método basado ...

Instalación de Python, paso a paso
Instalación de Python 3.6 A finales de 2016 se produjo el lanzamiento de Python 3.6 . El propósito de esta entrada es mostrar, pas...

Añadir, consultar, modificar y suprimir elementos en Numpy
Acceder a los elementos de un array. [], [,], ... Acceder a un elemento de un array. Para acceder a un elemento se utiliz...

Variables de control en Tkinter
Variables de control Las variables de control son objetos especiales que se asocian a los widgets para almacenar sus valore...

Cálculo con arrays Numpy
Numpy ofrece todo lo necesario para obtener un buen rendimiento cuando se trata de hacer cálculos con arrays. Por como está concebid...

Tkinter: interfaces gráficas en Python
Introducción Con Python hay muchas posibilidades para programar una interfaz gráfica de usuario (GUI) pero Tkinter es fácil d...

Operaciones con fechas y horas. Calendarios
Los módulos datetime y calendar amplían las posibilidades del módulo time que provee funciones para manipular expresiones de ti...

Convertir, copiar, ordenar, unir y dividir arrays Numpy
Esta entrada trata sobre algunos métodos que se utilizan en Numpy para convertir listas en arrays y viceversa; para copiar arrays d...

Tkinter: Tipos de ventanas
Ventanas de aplicación y de diálogos En la entrada anterior tratamos los distintos gestores de geometría que se utilizan para di...

El módulo random
El módulo random de la librería estándar de Python incluye un conjunto de funciones

```
)


Color = Enum(
    value='Color',
    names=[
        ('ROJO', '#FF0000'),
        ('AMARILLO', '#FFFF00'),
        ('VERDE', '#008000'),
    ],
)
```

Las enumeraciones de los ejemplos muestran que existen dos modos de declararlas: como una clase un tanto especial o con el método **Enum()** en una simple asignación. También, puede observarse que las constantes de una misma enumeración pueden tener o no distintos valores y de diferentes tipos (int, str, tuple, dict y otros).

En la enumeración **Respuesta** las constantes **SI** y **NO** tienen distintos valores. Sin embargo, en **Estado** tanto la constante **DETENIDO** como **ENESPERA** tienen el mismo valor, **0**. Esto no es ningún inconveniente, lo que no se admite es tener dos constantes con la misma denominación en una enumeración. Más adelante mostraremos también cómo forzar que las constantes tengan valores únicos.

En la enumeración **Andalucia** cada constante tiene asignada una tupla con dos valores que hacen referencia al número de habitantes (en 2019) y a la superficie en kilómetros cuadrados de cada provincia andaluza.

En la enumeración **Semana** se obtienen las constantes que son abreviaturas de los días de la semana del argumento **names**. Además, como no se indican valores se les asignarán de forma automática los enteros del **1** al **7**.

Finalmente, en la enumeración **Color** las constantes se obtienen también del argumento **names** aunque en este caso se corresponde con una lista de tuplas con dos cadenas que hacen referencia al nombre de la constante y a su valor, un código de color en hexadecimal representado como una cadena.

La sintaxis utilizada para expresar cualquier constante facilita la legibilidad del código. Simplemente, hay que indicar en primer lugar el nombre de la enumeración, seguido de un punto y el nombre de la constante:

```
NombreEnumeración.CONSTANTE
```

Basándonos en esta notación podemos referirnos a alguna de las constantes declaradas en los ejemplos anteriores del siguiente modo:

```
Respuesta.SI
Semana.LU
Color.ROJO
```

Pero conozcamos más de cerca estos objetos y sus atributos:

```
# Obtener la clase de una enumeración:

type(Respuesta)  # 'enum.EnumMeta'
type(Estado)  # 'enum.EnumMeta'

# Obtener el tipo de objeto de una constante:

type(Respuesta.SI)  # enum 'Respuesta'
type(Semana.JU)  # enum 'Semana'

# Obtener el nombre de una constante

respnom = Respuesta.SI.name  # 'SI'
estnom = Estado.DETENIDO.name  # 'DETENIDO'

# Obtener el valor de una constante

respval = Respuesta.SI.value  # 1
hab, sup = Andalucia.AL.value  # hab=715993; sup=8775
diaval = Semana.DO.value  # 7
color = Color.VERDE.value  # '#008000'

# Obtener una constante por su nombre

objnom1 = Respuesta['SI']  # Respuesta.SI: 1
```

que permiten obtener de distintos modos números a...

Archivo

marzo 2020 (1)

python.org

pypi.org

Sitios

- ActivePython
- Anaconda
- Bpython
- Django
- Flask
- Ipython
- IronPython
- Matplotlib
- MicroPython
- Numpy
- Pandas
- Pillow
- PortablePython
- PyBrain
- PyCharm
- PyDev
- PyGame
- Pypi
- PyPy
- Pyramid
- Python.org
- PyTorch
- SciPy.org
- Spyder
- Tensorflow
- TurboGears

```
objnom2 = Semana['LU']  # Semana.LU: 1

# Obtener la primera constante que tiene un valor determinado

objconst1 = Respuesta(2)  # Respuesta.NO: 2
objconst2 = Estado(0)  # Estado.DETENIDO: 0

# Comprobar si una constante pertenece a una enumeración

isinstance(Respuesta.SI, Respuesta)  # True
isinstance(Semana.DO, Semana)  # True

# Imprimir el objeto (No se obtiene el valor)

print(Respuesta.SI)  #  Respuesta.SI
print(Color.AMARILLO)  # Color.AMARILLO

# Imprimir la cadena de representación de una constante

print(repr(Respuesta.SI))  # Respuesta.SI: 1
print(repr(Estado.DETENIDO))  # Estado.DETENIDO: 0

# Comparar constantes con is o is not

Respuesta.SI is not Respuesta.NO  # True
Estado.DETENIDO is Estado.ENESPERA  # True

# Comparar constates con == o !=

Respuesta.SI != Respuesta.NO  # True
Estado.DETENIDO == Estado.ENESPERA  # True
```

Comparaciones. La clase IntEnum

Algo que no agrada a muchos es no poder comparar una constante con un valor entero determinado; pero esto es normal porque la referencia a una constante devuelve el objeto enumeración al que pertenece no un valor entero, una cadena u otro tipo de objeto. Mucho cuidado, cualquier comparación que se haga de este tipo siempre devolverá **False**:

```
Estado.DETENIDO == 0  # False
Respuesta.SI == 1  # False
```

Para este tipo de comparaciones podemos utilizar el atributo **value** que ya ha aparecido antes en los ejemplos:

```
Estado.DETENIDO.value == 0  # True
Respuesta.SI.value == 1  # True
```

No obstante, la clase **IntEnum** nació para permitir hacer este tipo comparaciones de forma más abreviada y directa, sin obligar a referenciar el atributo **value**. Para ello, tan solo tendremos que declarar las enumeraciones con valores basados en enteros del siguiente modo:

```
from enum import IntEnum


class Respuesta(IntEnum):
    SI = 1
    NO = 2


class Estado(IntEnum):
    DETENIDO = 0
    ENESPERA = 0
    INICIADO = 1


# Ahora es posible evaluar de forma más óptima
# el valor (entero) de una constante.

Estado.DETENIDO == 0  # True
Respuesta.SI == 1  # True
```

Iteraciones

Otra característica que tienen las enumeraciones, además de permitir comparaciones, consiste en que son objetos iterables. Esto significa que se pueden recorrer, una a una, todas sus constantes en el orden en que fueron declaradas:

```
for dia in Semana:
    print(dia.name, '->', dia.value)


# LU -> 1
# MA -> 2
# MI -> 3
# JU -> 4
# VI -> 5
# SA -> 6
# DO -> 7
```

Enumeraciones con constantes con valores únicos

Como se ha comentado antes, la enumeración **Estado** cuenta con las constantes **DETENIDO** y **ENESPERA** que tienen el mismo valor, **0**. Si queremos exigir que todas las constantes tengan valores únicos tendremos que agregar el decorador **@unique** a la declaración de la enumeración. Si agregamos el decorador y existen valores repetidos se producirá una excepción del tipo **ValueError**:

```
from enum import Enum, unique

@unique
class Estado(Enum):
    DETENIDO = 0
    ENESPERA = 0
    INICIADO = 1


# ValueError: duplicate values found in : ENESPERA -> DETENIDO
```

Enumeraciones con métodos

Las enumeraciones son clases de Python y como tales permiten incorporar métodos, algunos especiales, que amplían sobremanera el potencial de este tipo de objetos. En el siguiente ejemplo se declara la enumeración **Andalucia** con varios métodos para obtener los valores de habitantes y superficie de las constantes en una tupla, como valores independientes o bien devolviendo el cálculo de la densidad de población.

```
from enum import Enum


class Andalucia(Enum):
    AL = (715993, 8775)
    CA = (1240020, 7436)
    CO = (782516, 13771)
    GR = (914428, 12647)
    HU = (521428, 10128)
    JA = (633120, 13496)
    MA = (1660693, 7308)
    SE = (1941804, 14036)

    def __init__(self, hab, sup):
        self.hab = hab  # Núm. habitantes
        self.sup = sup  # En Km cuadrados

    @property
    def habitantes(self):
        return self.hab

    @property
    def superficie(self):
        return self.sup

    @property
    def densidad(self):
        return self.hab / self.sup


# Obtener información de la provincia de Córdoba (CO):
```

```
Andalucia.CO.value  # (782516, 13771)
Andalucia.CO.habitantes  # 782516
Andalucia.CO.superficie  # 13771
Andalucia.CO.densidad  # 56.823469610050104


# Obtener información de la provincia de Sevilla (SE):

Andalucia.SE.value   # (1941804, 14036)
Andalucia.SE.habitantes  # 1941804
Andalucia.SE.superficie  # 14036
Andalucia.SE.densidad  # 138.3445426047307
```

Relacionado:

- Palabras reservadas. Variables. Cadenas
- Expresiones de asignación
- Variables locales y variables globales
- Copia superficial y profunda de variables
- Diccionario de variables locales y globales
- Programación Orientada a Objetos (I)
- Programación Orientada a Objetos (II)
- Programación Orientada a Objetos (y III)

Publicado por Pherkad en 9:44

Etiquetas: Python

Entrada más reciente Inicio Entrada antigua