prog_datasci_3_apython_entrega

October 17, 2019

1 Programación para *Data Science*

1.1 Unidad 3: Estructuras de control y funciones en Python

En este Notebook encontraréis dos conjuntos de ejercicios: un primer conjunto de **ejercicios para practicar** y un segundo conjunto de **actividades evaluables** como PEC de la asignatura.

En cuanto al conjunto de ejercicios para practicar, éstos no puntúan para la PEC, pero os recomendamos que los intentéis resolver como parte del proceso de aprendizaje. Encontraréis ejemplos de posibles soluciones a los ejercicios al propio notebook, pero es importante que intentéis resolverlos vosotros antes de consultar las soluciones. Las soluciones os permitirán validar vuestras respuestas, así como ver alternativas de resolución de las actividades. También os animamos a preguntar cualquier duda que surja sobre la resolución de los **ejercicios para practicar** en el foro del aula.

En relación a las actividades evaluables, veréis que cada una de ellas tiene asociada una puntuación que indica el peso que tiene la actividad sobre la nota de la PEC. Adicionalmente, hay un ejercicio opcional, que no tiene puntuación dentro de la PEC, pero que se valora al final del semestre de cara a conceder las matrículas de honor y redondear las notas finales. Podéis sacar la máxima nota de la PEC sin necesidad de hacer este ejercicio! El objetivo de este ejercicio es que sirva como pequeño reto para los estudiantes que quieran profundizar en el contenido de la asignatura.

Además, veréis que todas las actividades tienen una etiqueta que indica los recursos necesarios para llevarla a cabo. Hay tres posibles etiquetas:

- NM **Sólo materiales**: las herramientas necesarias para realizar la actividad se pueden encontrar en los materiales de la asignatura.
- EG **Consulta externa guiada**: la actividad puede requerir hacer uso de herramientas que no se encuentran en los materiales de la asignatura, pero el enunciado contiene indicaciones de dónde o cómo encontrar la información adicional necesaria para resolver la actividad.
- EI **Consulta externa independente**: la actividad puede requerir hacer uso de herramientas que no se encuentran en los materiales de la asignatura, y el enunciado puede no incluir la descripción de dónde o cómo encontrar esta información adicional. Será necesario que el estudiante busque esta información utilizando los recursos que se han explicado en la asignatura.

Es importante notar que estas etiquetas no indican el nivel de dificultad del ejercicio, sino únicamente la necesidad de consulta de documentación externa para su resolución. Además, recordad que las **etiquetas son informativas**, pero podréis consultar referencias externas en cualquier

momento (aunque no se indique explícitamente) o puede ser que podáis hacer una actividad sin consultar ningún tipo de documentación. Por ejemplo, para resolver una actividad que sólo requiera los materiales de la asignatura, puedéis consultar referencias externas si queréis, ya sea tanto para ayudaros en la resolución como para ampliar el conocimiento!

En cuanto a la consulta de documentación externa en la resolución de los ejercicios, recordad **citar siempre la bibliografía utilizada** para resolver cada actividad.

1.2 Ejercicios para practicar

Los siguientes 3 ejercicios no puntúan para la PEC, pero os recomendamos que los intentéis resolver antes de pasar a los ejercicios propios de la PEC. También encontraréis las soluciones a estos ejercicios al final del Notebook.

1.2.1 Ejercicio 1

El ejercicio siguiente consiste en pasar un número en base 16 (hexadecimal, 0-9 / A-F) a base 10 (decimal). Para ello, debéis crear una **función** que dado un *string* que representa un número en hexadecimal, por ejemplo, `AE3F`, devuelva el número natural correspondiente. En este caso el resultado sería `44607`. NM

[1]:
```
# Respuesta
```

1.2.2 Ejercicio 2

Escribir una función que dado un número entero positivo, `N`, genere un archivo con el nombre`output.txt` que contendrá `N` líneas, donde cada línea deberá mostrar un número consecutivo de letras`A`. NM

Por ejemplo, si `N = 4`, el archivo generado deberá tener el siguiente contenido:

```
A
AA
AAA
AAAA
```

[2]:
```
# Respuesta
```

1.2.3 Ejercicio 3

Completad el código necesario para calcular el número de vocales y de consonantes respectivamente de un texto. NM

[3]:
```
def contar_vocales_y_consonantes(texto):
    # Cuenta las vocales contenidas en el string texto y también las␣
 ↪consonantes.
    num_vocales = 0
    num_consonantes = 0

    # Código que hay que completar.
```

```
    return (num_vocales, num_consonantes)
```

[4]:
```
# Respuesta
```

1.3 Ejercicios y preguntas teóricas para la PEC

A continuación, encontraréis los **ejercicios y preguntas teóricas que debéis completar en esta PEC** y que forman parte de la evaluación de esta unidad.

1.3.1 Ejercicio 1

Un peatón quiere cruzar una calle por un paso de peatones que tiene un semáforo. El peatón deberá decidir si cruza o no la calle, y lo hará considerando algunos factores. En concreto, el peatón cruzará la calle si el semáforo del paso de peatones está en verde y no hay ningún vehículo interfiriendo el paso. En caso contrario (es decir, si no se cumplen todas las condiciones anteriores), el peatón se quedará quieto y no cruzará la calle.

Escribir código que compruebe si el peatón puede o no cruzar la calle, y muestre por pantalla el mensaje "El peatón cruza" o "El peatón se queda quieto" dependiendo de la decisión del peatón.

Para comprobar las condiciones que afectan a la decisión del peatón, disponéis de las siguientes variables: * `color_semaforo`: una cadena de caracteres que puede contener los valores "verde", "amarillo" y "rojo" * `vehiculo_interfiriendo`: un booleano indicando si hay o no un vehículo interfiriendo (puede tomar los valores True o False)

(1.5 puntos) NM

[7]:
```
# Valores iniciales de las variables color_semaforo y vehiculo_interfiriendo
# Recordad que podéis cambiar estos valores para hacer pruebas con vuestro␣
 ↪código!

color_semaforo = "amarillo"
vehiculo_interfiriendo = False

# Respuesta
```

1.3.2 Ejercicio 2

Escribir una función que encapsule el comportamiento del código del ejercicio anterior. La función recibirá dos parámetros, el color del semáforo y si hay un vehículo interfiriendo y **devolverá un valor booleano** indicando si el peatón puede o no cruzar la calle. Adicionalmente, la función mostrará por pantalla el mismo mensaje que el ejercicio anterior.

Haced tres llamadas de ejemplo a la función anterior, con valores diferentes de los parámetros de entrada, de forma que cada llamada devuelva uno de los tres posibles valores de retorno (`True`,`False`, `-1`). Comprobad que el resultado devuelto por la función es el que se esperaba.

Consideraciones:

- La función debe controlar que los valores de los parámetros son válidos. En caso contrario, mostrará un mensaje de error ("Los parámetros introducidos no son válidos") y devolverá el

valor -1. Los parámetros se consideran válidos si son del tipo correcto (cadena de caracteres para el color del semáforo y booleano para el flag de vehículo) y contienen valores válidos (detallados en el ejercicio anterior).

(1.5 puntos) NM

[6]:
```
# Respuesta: Definición de la función
```

[8]:
```
# Respuesta: Llamadas a la función
```

1.3.3 Ejercicio 3

Escribid una función que dado un número entero positivo, `N`, genere un archivo con el nombre `output.txt` que contendrá $2N - 1$ líneas, siguiendo un patrón de "media pajarita".

Por ejemplo, si $N = 3$, el archivo generado deberá tener el siguiente contenido:

```
***
**
*
**
***
```

o si $N = 4$, el siguiente contenido:

```
****
***
**
*
**
***
****
```

Nota: `N` siempre será ≥ 1. Si $N = 1$, el fichero contendrá una única línea con un solo asterisco.

(1.5 puntos) NM

[8]:
```
# Respuesta
```

1.3.4 Ejercicio 4

Escribid una función que muestre por pantalla todas las posibles palabras de tres letras que se pueden hacer con las letras "A", "B" y "C". La función deberá devolver una lista con estas palabras.**(1.5 puntos)** NM

[9]:
```
letras = ["A", "B", "C"]

# Respuesta
```

1.3.5 Ejercicio 5

Escribid una función que dado un diccionario con la masa de los planetas del sistema solar, pregunte al usuario que introduzca el nombre de un planeta y muestre por pantalla el nombre del planeta y su masa. La función devolverá la masa del planeta.

Consideraciones:

- La función debe controlar que el valor introducido por el usuario es un nombre de un planeta. En caso contrario, mostrará un mensaje de error ("El nombre introducido no corresponde a ningún planeta") y devolverá el valor -1.
- Si el usuario introduce "Pluton", hay que mostrar un mensaje que informe al usuario que desde 2006, Plutón ya no es considerado un planeta, y devolver -1.
- Debéis tener en cuenta que el nombre de los planetas que nos pasan por parámetro puede ser en minúsculas, mayúsculas o una combinación de ambas.
- Podéis asumir que no habrá acentos en los nombres de los planetas.

Nota 1: Para realizar la actividad, tendréis que capturar un texto que entrará el usuario. Consultad la documentación oficial de la función input para ver cómo hacerlo.

Nota 2: También tendréis que pensar cómo tratar el hecho de que el usuario pueda utilizar mayúsculas y minúsculas en la escritura del nombre del planeta. Os animamos a usar un buscador para intentar encontrar alguna alternativa para resolver este subproblema! Recordad citar las referencias que hayáis usado para resolverlo!

(1.5 puntos) EG

[10]:
```
masas = {'Mercurio': 3.20*10**23, 'Venus': 4.87*10**24, 'Terra': 5.9*10**24,␣
 ↪'Marte': 6.42*10**23, 'Jupiter': 1.90*10**27,
          'Saturno': 5.68*10**26, 'Urano': 8.68*10**25, 'Neptuno': 1.02*10**26}

# Respuesta
```

Referencias consultadas:

Incluir aquí las referencias

1.3.6 Ejercicio 6

Las excepciones son errores detectados en tiempo de ejecución. Pueden y deben ser gestionadas por el programador para minimizar el riesgo de que un determinado programa falle de manera no controlada. Escribid, en lenguaje Python, como generar y capturar la siguiente excepción: `NameError`. **(1 punto)** EG

Nota: Para hacer este ejercicio, consultad la documentación oficial de Python sobre gestión de errores.

[11]:
```
# Respuesta
```

1.3.7 Ejercicio 7

Python dispone de un *idiom* muy útil conocido como *list comprehensions*. Utilizando este *idiom*, proporcionad una expresión que devuelva las listas siguientes.

Nota: Para realizar esta actividad necesitaréis investigar qué son las *list comprehensions* y qué sintaxis utilizan. Para ello, se recomienda en primer lugar que utilicéis un buscador para encontrar

información genérica sobre esta construcción. Después, os recomendamos que consultéis stackoverflow para ver algunos ejemplos de problemas que se pueden resolver con esta construcción.

stackoverflow es un sitio de preguntas-y-respuestas muy popular entre programadores. Veréis que para la gran mayoría de las dudas que tengáis, habrá alguien que ya les habrá tenido (y consultado) anteriormente! Así pues, más allá de preguntar vosotros mismos las dudas allí (nosotros ya tenemos el foro del aula para ello!), consultar esta web os permitirá ver qué soluciones proponen otros programadores a estas dudas. A menudo habrá más de una solución a un mismo problema, y podréis valorar cuál es la más adecuada para vuestro problema.

Para ver ejemplos de problemas que son adecuados para resolver con *list comprehensions*, os recomendamos leer las siguientes páginas: * https://stackoverflow.com/questions/12555443/squaring-all-elements-in-a-list * https://stackoverflow.com/questions/18551458/how-to-frame-two-for-loops-in-list-comprehension-python * https://stackoverflow.com/questions/24442091/list-comprehension-with-condition * https://stackoverflow.com/questions/41676212/i-want-to-return-only-the-odd-numbers-in-a-list * https://stackoverflow.com/questions/4260280/if-else-in-a-list-comprehension

(1.5 puntos) EG

a) Una lista de los cubos de los números de la lista `list_1`:

[12]:
```
list_1 = [1, 2, 3, 4, 5, 6, 7, 8, 9]

# Respuesta
```

b) Una lista de los cuadrados de los números impares de la lista `list_1`:

[13]:
```
# Respuesta
```

c) Una lista con las tuplas `(0, 0)`, `(0, 1)`, `(0, 2)`, `(0, 3)`, `(1, 0)`, `(1, 1)`, `(1, 2)`, `(1, 3)`, `(2, 0)`, `(2, 1)`, `(2, 2)`, `(2, 3)`, `(3, 0)`, `(3, 1)`, `(3, 2)`, `(3, 3)` a partir de las listas `list_2` i `list_3`:

[15]:
```
list_2 = range(4)
list_3 = range(4)

# Respuesta
```

[15]:
```
[(0, 0),
 (0, 1),
 (0, 2),
 (0, 3),
 (1, 0),
 (1, 1),
 (1, 2),
 (1, 3),
 (2, 0),
 (2, 1),
 (2, 2),
```

```
 (2, 3),
 (3, 0),
 (3, 1),
 (3, 2),
 (3, 3)]
```

1.3.8 Ejercicio Opcional

Al final de la Edad Media, en Francia, el diplomático francés Blaise de Vigenère desarrolló un algoritmo para cifrar mensajes que nadie fue capaz de romper durante aproximadamente 250 años. Este algoritmo se conoce con el nombre de cifrado de Vigenère.

El cifrado de Vigenère consiste en añadir a cada una de las letras de un texto un desplazamiento a partir de una clave secreta para conseguir una nueva letra diferente del original. Veamos un ejemplo:

Si asignamos el número 1 en la primera letra del abecedario, A, 2 a la siguiente, B, etc., imaginad que tenemos el siguiente mensaje: ABC 123

y la siguiente clave secreta: DEF 456

A cada letra del mensaje original le aplicamos un desplazamiento en función de la misma posición dentro de la clave secreta. Por lo tanto, el mensaje cifrado quedaría de la siguiente forma: E G I (1 + 4) (2 + 5) (3 + 6)

Escribid una función que, dado un mensaje y una clave secreta, calcule y devuelva el mensaje cifrado.

Consideraciones:

- Utilizad como alfabeto de entrada **el alfabeto inglés en mayúsculas**.
- El valor predeterminado de la clave secreta será **DATASCI**.

EI

[14]:
```python
def cifrado_vigenere(mensaje, llave="DATASCI"):
    """
    Cifrar mensaje utilizando el cifrado de Vigenère
    """
    mensaje_cifrado = ""

    # Código que hay que completar


    return mensaje_cifrado

# Aquí puede añadir más ejemplos:
print(cifrado_vigenere("ATACAREMOS POR LA MANANA"))
```

1.4 Soluciones ejercicios para practicar

1.4.1 Ejercicio 1

El ejercicio siguiente consiste en pasar un número en base 16 (hexadecimal, 0-9 / A-F) a base 10 (decimal). Para ello, debe crear una **función** que dado un *string* que representa un número en hexadecimal, por ejemplo, `AE3F`, devuelva el número natural correspondiente. En este caso el resultado sería 44607.

Respuesta

La formula para convertir un número hexadecimal a un número decimal, tomando como ejemplo el número AE3F, es: `A * 16**3 + E * 16**2 + 3 * 16**1 + F * 16**0 = 10 * 16**3 + 14 * 16**2 + 3 * 16**1 + 15 * 16**0`

[6]:
```
def hex_to_dec(numero_hexadecimal):

    hex_mapping = {'0': 0, '1': 1, '2': 2, '3': 3, '4': 4, '5': 5,
                   '6': 6, '7': 7, '8': 8, '9': 9, 'A': 10, 'B': 11,
                   'C': 12, 'D': 13, 'E': 14, 'F': 15}

    # Primero, comprobamos que el número que se pasa por parámetro es␣
↪hexadecimal
    all_hex = True
    for c in numero_hexadecimal:
        if not c in hex_mapping.keys():
            all_hex = False

    if all_hex:

        # Definimos la base para realizar las operaciones
        base = 16
        numero_decimal = 0

        # Invertimos el número hexadecimal para que nos sea más fácil trabajar␣
↪con los índices
        numero_hexadecimal = numero_hexadecimal[::-1]

        for i in range(len(numero_hexadecimal)):
            # Por cada carácter hexadecimal, aplicamos la formula c * base **␣
↪y,
            # donde c es la representación decimal del carácter y
            # sumamos el resultado al resultado obtenido en la iteracó anterior
            numero_decimal = numero_decimal +␣
↪hex_mapping[numero_hexadecimal[i]] * base**i

        return numero_decimal
    else:
        print("El número introducido no es correcto")

print(hex_to_dec('AE3F'))
```

```
print(hex_to_dec('FFF'))
print(hex_to_dec('123'))
```

```
44607
4095
291
```

1.4.2 Ejercicio 2

Escribid una función que dado un número entero positivo, `N`, genere un archivo con el nombre`output.txt` que contendrá `N` líneas, donde cada línea deberá mostrar un número consecutivo de letras`A`.

Por ejemplo, si `N = 4`, el archivo generado deberá tener el siguiente contenido:

```
A
AA
AAA
AAAA
```

Respuesta

[18]:
```
# Definimos una función que recibirá un número entero por parámetro
def generar_archivo(N):
    # Abrimos el archivo output.txt en modo escritura.
    # El parámetro 'w' hará que si el archivo existe, éste se sobrescribirá
    with open('output.txt', 'w') as fd:

        # Recorremos el rango [0 ... N-1]
        for i in range(N):
            # Generamos una cadena con un número ascendente de caracteres A
            linea = 'A' * (i + 1)
            # Escribimos cada cadena generada en el fichero, añadiendo un salto␣
 ↪de línea
            fd.write(linea + '\n')

generar_archivo(4)
```

1.4.3 Ejercicio 3

Completad el código necesario para calcular el número de vocales y de consonantes respectivamente de un texto.

Respuesta

[19]:
```
def contar_vocales_y_consonantes(text):
    # Cuenta las vocales contenidas en el string texto y también las␣
 ↪consonantes.
    num_vocales = 0
    num_consonantes = 0
```

```
    # Definimos una lista con las vocales
    vocales = ['a', 'e', 'i', 'o', 'u']

    for c in text.lower(): # Podemos convertir el texto en minúsculas para␣
 ↪simplificar los cálculos
        if c in vocales:
            num_vocales = num_vocales + 1
        elif c > 'a' and c <= 'z':
            num_consonantes = num_consonantes + 1

    return (num_vocales, num_consonantes)

texto = "Orbiting Earth in the spaceship, I saw how beautiful our planet is. \
            People, let us preserve and increase this beauty, not destroy it!"

num_vocales, num_consonantes = contar_vocales_y_consonantes(texto)
print( "El numero de vocales és %d." % num_vocales)
print( "El numero de consonantes és %d." % num_consonantes)
```

```
El numero de vocales és 44.
El numero de consonantes és 62.
```