prog_datasci_3_apython_entrega

March 3, 2021

1 Fundamentos de Programación

1.1 PAC 3 - Enunciado

En este Notebook se encontraréis el conjunto de actividades evaluables como PEC de la asignatura. Veréis que cada una de ellas tiene asociada una puntuación, que indica el peso que tiene la actividad sobre la nota final 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. Podréis sacar la máxima nota de la PAC 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.

Veréis que todas las actividades de la PEC 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 independiente**: 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 y preguntas teóricas para la PEC

1.2.1 Ejercicio 1

Tenemos un problema. Hemos perdido parte del código del ejercicio, pero sabemos que se han utilizado al menos dos variables: lst y suma. ¿Qué valor tendría la variable suma en cada uno de los siguientes casos? Explica lo que sucede en cada fragmento de código teniendo en cuenta que para todos los supuestos, el estado inicial de dichas variables viene dado siempre por el fragmento de código siguiente y sus resultados por pantalla:

```
<div style="float: left; width: 100%;">
   <img src="./data/enunciado_ej1.png", align="left">
</div>
```

Recuerda que siempre puedes testear tus suposiciones, construyéndote las variables lst y suma que verifiquen las condiciones iniciales del enunciado. EG **(2.5 puntos)**

NOTA: En los enlances siguientes encontraréis información adicional para poder interpretar las instrucciones 'break' y 'continue': https://stackoverflow.com/questions/20893641/why-is-continue-not-working/20893653#20893653

(a)

```
for n in range(1, len(lst)):
    continue
    suma += n**2
```

(b)

```
for val in range(len(lst) + 10):
    suma += val
    break
```

(c)

```
l = len(lst)
while((l > 2) and (suma < 100)):
    suma += 1
```

(d)

```
c1 = suma < 2
c2 = True
condiciones = c1 or (not c2)

while(condiciones):
    suma = 10
    if c1 and suma > 2:
        break
```

(e)

```
l = []
for k in range(1, 3):
    l.append(len(lst) < k)
```

```
suma = sum(l)
```

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

1.2.2 Ejercicio 2

(a) El fichero filename contiene información sobre todos los modelos de coche del mercado. Cada una de las columnas de información disponibles en dicho fichero están separadas entre ellas con un punto y coma. La primera fila contiene la cabecera, que define cada una de las columnas, y la primera columna define el tipo de vehículo ('suv', 'berlina', 'monovolumen', etc.). Con estos datos, explica qué miden las variables total y suv y razona por qué la variable total se inicializa a -1. NM **(0.5 puntos)**

```
total = -1
suv = 0

with open(filename) as f:
    for line in f:
        if line.split(";")[0] == 'suv':
            suv += 1
        total += 1
```

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

(b) Asigna un valor cualquiera a las variables total y suv del apartado anterior y escríbelas en un fichero "output.csv" en la carpeta "data" del directorio de trabajo. Los dos valores deberán escribirse en la misma fila, separados por una coma. No olvides añadir una cabecera para las dos columnas. Acto seguido, abre dicho fichero e imprime por pantalla los valores escritos previamente. NM **(1.0 puntos)**

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

1.2.3 Ejercicio 3

(a) Utiliza la librería **sys** para imprimir por pantalla una **lista** con las distintas rutas que contiene la variable **path** del sistema. NM **(0.5 puntos)**

(b) Recorre la lista anterior mediante un bucle e imprime por pantalla el índice de cada posición y la longitud (número de caracteres) de cada una de las rutas. NM **(0.5 puntos)**

(c) Construye un diccionario que tenga como "keys" el índice de la posición de cada ruta dentro de la lista y como "values" una tupla donde la primera posición sea la ruta (cadena de caracteres) y la segunda posición corresponda a la longitud (número de caracteres) de la ruta correspondiente. NM **(0.5 puntos)**

(d) Define una función que tome como argumento el diccionario del apartado anterior e imprima por pantalla el "key" y el "value" para cada uno de sus elementos siguiendo el formato siguiente: "key" : "value". No es necesario que retorne ningún valor. NM **(0.5 puntos)**

(e) Define otra función que tome nuevamente como argumento el diccionario del apartado (c) y que retorne el índice (es decir el "key" del diccionario) correspondiente al de la ruta más larga (si hay más de una con la misma longitud, devuelve la primera que encuentres). NM **(0.5 puntos)**

(f) Utiliza la función anterior para identificar la ruta más larga y construye una lista que solo contega las vocales de dicha ruta. NM **(0.5 puntos)**

[3]:
```
## Respuesta
```

1.2.4 Ejercicio 4

Python dispone de un **idiom** muy útil conocido como **list comprehension** y esto es lo que vamos a trabajar en este ejercicio a partir de la lista siguiente:

[1]:
```
collection = ['error', 10, [], 2e+01, {"OK - NOOK"}, 3.5, True, {}, 3, 5j, 14,␣
↪7]
```

NOTA: Para realizar esta actividad necesitaréis investigar qué son las list comprehension 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 (un sitio de preguntas-y-respuestas muy popular entre programadores) para ver algunos ejemplos de problemas que se pueden resolver con esta construcción: * https://stackoverflow.com/questions/12555443/squaring-all-elements-in-a-list * https://stackoverflow.com/questions/57166908/using-list-comprehension-i-want-to-print-odd-even-with-string-indicating-even/57167016

(a) Recorre la lista e imprime por pantalla cada elemento y la clase/tipo a la que pertenecen. En este apartado no es necesario el uso de list comprehension. NM **(0.5 puntos)**

(b) Construye una lista idéntica a la del enunciado pero recorriendo la lista collection con una sola expresión mediante list comprehension, luego imprímela por pantalla. EG **(0.5 puntos)**

(c) Utiliza list comprehension para reconstruir la misma lista pero esta vez, reemplazando los valores que no sean de la clase/tipo int por el valor especial nan (Not a Number) que puedes obtener mediante la función float() (https://www.programiz.com/python-programming/methods/built-in/float). EG **(1.0 punto)**

(d) Define una función que reciba como primer argumento la lista generada en el apartado anterior y que retorne, según el valor de un segundo argumento de tipo booleano, o una lista con los valores enteros (es decir, sin los nan) o bien, la suma de dichos valores enteros. Por defecto, si solo se pasa un argumento, se retornará la suma. Los dos valores de retorno posibles (la suma o la lista) se obtendrán mediante expresiones compactas que incluyan el uso de list comprehension. La función definida se testeará en ambos casos y el valor retornado se imprimirá por pantalla. Para identificar si un elemento es nan existen varias opciones que podréis encontrar en el siguiente enlance: https://stackoverflow.com/questions/944700/how-can-i-check-for-nan-values. EG **(1.0 punto)**

(e) De manera análoga, también existen los dict comprehension. Construye un diccionario siguiendo esta funcionalidad de Python, donde el "key" corresponda al índice de la posición de

cada elemento dentro de la lista definida en el enunciado y el “value”, a su clase/tipo. EI **(Opcional)**

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