prog_datasci_3_python_entrega_cast

March 22, 2023

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 para la PEC

A continuación encontraréis los ejercicios que se deben completar en esta PEC y que forman parte de la evaluación de esta unidad.

1.2.1 Ejercicio 1

Considera la lista siguiente **(2 puntos)**:

[ ]:
```
lista=[20, 'Hola', True, 'Python', -3]
```

a) Tienes un fragmento de código. Modifícalo para que muestre los elementos que se indican a continuación. Razona tu respuesta NM **(0.5 puntos)**

Nota: Tienes que realizar los mínimos cambios posibles.

a1) Debe mostrar los siguientes elementos: Hola, True, Python, -3.

a2) Debe mostrar los siguientes elementos: 20, Hola, True.

[ ]:
```
# Código a modificar
for n in lista:
    if n == 'Hola':
        continue
    print(n)
```

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

b) Escribe un código que, dada la lista que estamos trabajando, 1) detecte los elementos de tipo string de la lista, 2) los ordene al revés (es decir, Hola sería aloH y Python, nohtyP) y 3) los muestre por pantalla. NM **(0.5 puntos)**

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

c) Escribe un código que, dado un rango de números de 1 a 25, guarde los números primos en una lista. Muestra después por pantalla los elementos de la lista.EG **(1 punto)**

Nota: Puedes consultar cómo detectar si un número es primero en este post de stack overflow.

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

1.2.2 Ejercicio 2

En este ejercicio vamos a crear una calculadora de números primos (**2.75 puntos**):

a) Pon el código del ejercicio 1c) en forma de función. Debe cumplir los siguientes puntos:

- Como entrada (*input*), especifica el número máximo que se desea recorrer (por ejemplo, en el ejercicio anterior, el número máximo era 25).
- Como salida (*output*), debe devolver la lista de números primos.

NM **(1 punto)**

```
## Respuesta
```

b) Utiliza los casos de prueba detallados en la tabla siguiente para comprobar que tu código funciona correctamente. NM **(0.25 puntos)**

máximo	lista resultado
9	[2, 3, 5, 7]
15	[2, 3, 5, 7, 11, 13]
50	[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]

```
## Respuesta
```

c) La calculadora funciona bien, pero nos hemos dado cuenta de que si el usuario introduce por error un valor que no sea un número, todo el programa falla. Para evitar esta situación, utiliza tu conocimiento sobre gestión de errores y excepciones para obtener el siguiente resultado:

- El usuario tiene que introducir el valor por pantalla. NM (**0.5 puntos**)
- Cuando el usuario introduzca un valor erróneo, el programa le avisará, en lugar de fallar directamente. NM (**0.5 puntos**)
- Una vez que el usuario ha sido avisado, el programa permite introducir el valor de nuevo una vez más. NM (**0.5 puntos**)

(**1.5 puntos**)

```
## Respuesta
```

1.2.3 Ejercicio 3

Tal y como hemos visto en la teoría, las *list comprehensions* son muy útiles para crear listas en una sola línea de código. (**1.75 puntos**)

a) A continuación, te proponemos que transformes los códigos siguientes a *list comprehensions*: NM (**1 punto**)

```
#a1)

result = []

for i in range(7):
        result.append(i**3+2*i)

print(result)
```

```
## Respuesta
```

[ ]:
```python
#a2)

list_elem2 = [16, 17, 7, 24, 5, 13, 100, 87]
new_list = []

for x in list_elem2:

    if x%2 == 0:
        new_list.append(x*2)
    else:
        new_list.append(x+2)

print(new_list)
```

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

b) Dada la lista siguiente: **(0.75 puntos)**

[ ]:
```python
lista=[10, "python", 3 , "UOC", True]
```

b1) Utiliza una *list comprehension* para detectar los strings de la lista y luego muestra el resultado por pantalla. NM **(0.5 puntos)**

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

b2) Usa los casos de prueba detallados en la tabla siguiente para comprobar que tu código funciona correctamente. NM **(0.25 puntos)**

lista	resultado
[‘hola’, False, 25, 41, True]	[‘hola’]
[“oso”, “perro”, “gato” , “conejo”, True]	[‘oso’, ‘perro’, ‘gato’, ‘conejo’]
[44, 7, 26, 14, 100]	[]

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

c) Hemos creado la siguiente list comprehension y nos hemos fijado en que como resultado tenemos una lista de `None`. Piensa y explica por qué obtenemos este resultado.(**Opcional**)

[ ]:
```python
#c)

# Creamos las listas
animales = ['gato', 'perro', 'raton','panda']
felino = ['gato']

# Mostramos el resultado por pantalla
[print(f"El {animal} es un felino") if animal in felino else print(f"El␣
 ↪{animal} no es un felino") for animal in animales]
```

1.2.4 Ejercicio 4

Mientras que las *list comprehensions* se utilizan para crear listas, la lambda es una función que se puede procesar como otras funciones y, por tanto, devolver valores o listas. En este ejercicio pondremos en práctica distintas funcionalidades de este tipo de funciones.

En los siguientes apartados, completa el código escribiendo la función lambda que realice la función indicada. Los guiones — indican dónde deberías poner la función lambda.

(**1.25 puntos**)

a) Dada la siguiente lista (list_str), escribe una función lambda que devuelva los elementos de la lista que acaban en "o". Muestra los elementos por pantalla. EG (**0.25 puntos**)

list_str = ["hello","hola","salut","ciao", "hallo", "hi", "konnichiwa"]

list(filter(lambda x: —, list_str))

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

b) Dada la lista del apartado a), escribe una función lambda que se quede con el elemento de la lista de mayor longitud (es decir, el string más largo). Muestra el elemento por pantalla.EG (**0.5 puntos**)

max_list = max(list_str, key = lambda x: —)

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

c) Escribe una expresión que, dada una lista de números, te cree otra lista con los elementos a la raiz quadrada. Puedes utilizar la función pow() para especificar el orden de la potencia (por ejemplo, pow(x, 0.5) hace la raiz cuadrada).

En este ejercicio no os damos el código para completar, sino que deberas escribir toda la expresión (no sólo la función lambda). Puede ser útil explorar la funcionalidad de map(). EG (**0.5 puntos**)

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

1.2.5 Ejercicio 5

Las funciones `range`, `ennumerate` y `zip` se utilizan mucho en la programación en Python. Por eso, en este ejercicio vamos a trabajar sus usos más habituales. **(1.25 puntos)**

a) Utiliza la función `range` para mostrar por pantalla los números impares entre 1 y 15 (ambos incluidos): NM (**0.25 puntos**)

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

b) Dada una lista, utiliza `ennumerate` para crear un diccionario que tenga como llave (key) el índice del elemento y, como valor, el elemento de la lista. En este caso, queremos que el índice no empiece con 0, sinó con 100. Es decir, el primero elemento seria 100: pan, el segundo, 101: leche, etc. EG (**0.5 puntos**)

Nota: Revisa la documentación de la función enumerate para tener más información.

[ ]:
```
# Creamos la lista
lista_compra = ['pan', 'leche', 'naranjas', 'galletas', 'patatas']
```

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

c) En este apartado estamos explorando al funcionalidad de zip(). Tenemos 3 listas de distintos tamaños, y queremos mostrar sus elementos. ¿Por qué se muestran 3 elementos de cada lista en el primer loop? ¿Por qué solo se muestran 2 elementos de cada lista en el segundo loop? Razona la respuesta. NM (**0.5 puntos**)

[ ]:
```
numbersList = [1, 2, 3]
str_list = ['one', 'two']
numbers_tuple = ('ONE', 'TWO', 'THREE', 'FOUR')

print('Primer loop: ')
for elem1, elem2 in zip(numbersList, numbers_tuple):
    print(elem1, elem2)

print('Segundo loop: ')
for elem1, elem2, elem3 in  zip(numbersList, str_list, numbers_tuple):
    print(elem1, elem2, elem3)
```

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

1.2.6 Ejercicio 6

En la carpeta `data` podéis encontrar el fichero `boxer_dataset.txt`, un dataset con información de boxeadores profesionales. La primera columna contiene la cabecera, que define cada una de las columnas (Ranking, Surname, Name, Country, Weight y Sex) y, a continuación, tenemos la información de los boxeadores ordenados por su ranking.

(1 punto)

a) Abre el archivo y responde a las siguientes preguntas: NM **(0.5 puntos)**

- ¿Cuántos **boxeadores españoles** hay?
- ¿Cuál es el porcentaje de españoles respecto al total de boxeadores? Muestra el resultado con **sólo 2 decimales**.

Nota: Los apartados a) y b) se tiene que responder utilizando la función `open()`.

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

b) Guarda, en un nuevo archivo de texto, sólo la información de los boxeadores que **pesan más de 150** (es decir, aquellas filas donde Weight sea mayor de 150). NM **(0.5 puntos)**

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

c) En teoría hemos visto también la librería pandas, que nos permite trabajar con archivos de manera muy sencilla. Utilízala para abrir el archivo de texto y muestra las 10 primeras líneas por pantalla. (**Opcional**)

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