prog_datasci_3_python_cast_entrega

October 16, 2022

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 siguiente lista de elementos **(1.5 puntos)**:

[1]:
```
list_elem = [1, 5, 10, "python", 33, True]
```

a) Lee el código que aparece a continuación y explica detalladamente en qué consiste y el resultado obtenido. NM (**0.25 puntos**)

Nota 1: Puedes consultar este enllace para saber más sobre los f-Strings, unas expresiones muy utilizadas con la función *print()*.

Nota 2: Puedes consultar este enllace para conocer la función `type()`.

[2]:
```
for n in list_elem:
    print(f"Element {n} is a ", type(n))
```

```
Element 1 is a  <class 'int'>
Element 5 is a  <class 'int'>
Element 10 is a  <class 'int'>
Element python is a  <class 'str'>
Element 33 is a  <class 'int'>
Element True is a  <class 'bool'>
```

Respuesta

b) Mientras estábamos trabajando, hemos perdido una parte del código. Tenemos el resultado que obteníamos, pero el código está incompleto y no funciona. A partir de la información que tenemos, reescribe las líneas de código que faltan para obtener el mismo resultado. NM (**0.5 puntos**)

[ ]:
```
#b1)

for n in list_elem:
    # Añade código

        print(f"Element {n} is a string")
    else:
        print(f"Element {n} is NOT a string")
```

[ ]:
```
# Resultado esperado

#Element 1 is NOT a string
#Element 5 is NOT a string
```

```
#Element 10 is NOT a string
#Element hello is a string
#Element 33 is NOT a string
#Element True is NOT a string
```

[ ]:
```
#b2)

for n in list_elem:
    # Añade código

        print(f"Element {n} is a string")
    # Añade código

        print(f"Element {n} is a Boolean")
    else:
        print(f"Element {n} is an integer")
```

[ ]:
```
# Resultado esperado

#Element 1 is an integer
#Element 5 is an integer
#Element 10 is an integer
#Element hello is a string
#Element 33 is an integer
#Element True is a Boolean
```

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

c) Escribe un código que, mediante `if`, muestre por pantalla sólo los números 10 y 33 de la lista **list_elem**. NM (**0.25 puntos**)

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

d) Qué diferencia existe entre el código d1) y el d2)? Ejecútalos y explica en detalle los resultados. NM (**0.5 puntos**)

[5]:
```
#d1)

# Recorremos los elementos de la lista
for n in list_elem:
    if n == 10:
        break
        # Mostramos el elemento por pantalla
    print(n)
```

```
1
5
```

[6]:
```
#d2)

# Recorremos los elementos de la lista
for n in list_elem:
    if n == 10:
        continue
        # Mostramos el elemento por pantalla
    print(n)
```

```
1
5
python
33
True
```

Respuesta

1.2.2 Ejercicio 2

a) Queremos hacer un programa que vaya sumando el dinero que gastamos y que, cuando lleguemos a un límite que definimos nosotros, nos avise y detenga el programa. El código debe tener las siguientes características:

- Empezaremos con un dinero inicial 0 y tendremos un límite de 50. NM (**0.25 puntos**)
- En cada iteración, incrementaremos el dinero que hemos gastado por un valor random entre 0 y 10 mediante la función randint. EG (**0.5 puntos**)
- En cada iteración, se tiene que mostrar por pantalla cuánto dinero hemos gastado y cuánto nos quedan. Es decir, la diferencia entre el límite y el dinero total gastado en cada iteración. NM (**0.25 puntos**)
- Una vez lleguemos o superemos el límite, se deberá avisar por pantalla y mostrar también cuánto dinero hemos gastado finalmente y cuánto nos hemos pasado del límite. NM (**0.5 puntos**)

(**1.5 puntos**)

[7]:
```
## Respuesta
```

b) El código anterior es muy interesante y queremos utilizarlo para apuntar lo que gastamos cuando vamos a comprar. Por este motivo, es conveniente ponerlo en forma de función, ya que nos permitirá interaccionar con él más fácilmente. Para ello, tendremos que tener en cuenta los siguientes puntos:

- La función deberá tener como parámetro de entrada el límite máximo que podemos gastar. NM (**0.25 puntos**)
- El usuario deberá introducir cada uno de los gastos mediante la función **input**, ya no será un valor aleatorio. NM (**0.5 puntos**)
- Tal y como se ha comentado en el apartado anterior, se deberá mostrar por pantalla cuánto dinero hemos gastado, lo que nos queda, y, si llegamos al límite, se deberá avisar al usuario. NM (**0.75 puntos**)

(**1.5 puntos**)

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

c) La calculadora de gastos funciona bien, pero nos hemos dado cuenta de que si el usuario introduce por error algo 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:

- 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. Si vuelve a introducir un valor erróneo, el programa se cierra. NM (**0.5 puntos**)

(**1 punto**)

[9]:
```
## Respuesta
```

1.2.3 Ejercicio 3

a) Tal y como hemos visto en la teoría, las *list comprehensions* son muy útiles para crear listas en una sola línea de código. A continuación, te proponemos que transformes los códigos siguientes a *list comprehensions*: NM

(**1.5 puntos**)

[10]:
```
#a1)

result = []

for i in range(5):
    result.append(i**3)

print(result)
```

```
[0, 1, 8, 27, 64]
```

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

[12]:
```
#a2)

list_elem2 = [10, 45, 57, 4, 2, 13, 98, 7]
new_list = []

for x in list_elem2:

    if x >= 45:
        new_list.append(x+1)
    else:
```

```
        new_list.append(x+5)

print(new_list)
```

```
[15, 46, 58, 9, 7, 18, 99, 12]
```

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

[14]:
```
#a3)

names = ['Alex','Peter','Anna','John','Felix','Pedro']
new_names = []

for name in names:
    if name.lower().startswith('a'):
        new_names.append(name)

print(new_names)
```

```
['Alex', 'Anna']
```

[15]:
```
## Respuesta
```

b) 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**)

[16]:
```
#b)

# Creamos las listas
days = ['Lunes','Martes','Miércoles','Jueves','Viernes','Sábado','Domingo']
weekend = ['Sábado','Domingo']

# Mostramos el resultado por pantalla
[print(f"Viva, ya es {d}") if d in weekend else print(f"Ànimos, aún es {d}")␣
 ↪for d in days]
```

```
Ànimos, aún es Lunes
Ànimos, aún es Martes
Ànimos, aún es Miércoles
Ànimos, aún es Jueves
Ànimos, aún es Viernes
Viva, ya es Sábado
Viva, ya es Domingo
```

[16]:
```
[None, None, None, None, None, None, None]
```

Respuesta

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.

a) Dada la siguiente lista (list_str), escribe una función lambda que devuelva los elementos de la lista que comienzan por "h". Muestra los elementos por pantalla.NM (**0.25 puntos**)

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

list(filter(lambda x: —, list_str))

b) Dada la lista del apartado b), escribe una función lambda que se quede con el elemento de la lista de menor longitud (es decir, el string más corto). Muestra el elemento por pantalla.NM (**0.25 puntos**)

min_list = min(list_str, key = lambda x: —)

c) Escribe una expresión que, dada una lista de números, te cree otra lista con los elementos elevados a la potencia de 5. Puedes utilizar la función pow() para especificar el orden de la potencia (por ejemplo, pow(x, 2) eleva x al cuadrado).

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**)

(**1 punto**)

[17]:
```
## Respuesta
```

1.2.5 Ejercicio 5

a) Utiliza la función `range` para mostrar por pantalla los números pares entre 2 y 10 (ambos incluidos): NM (**0.25 puntos**)

[18]:
```
## Respuesta
```

b) Crea una función que, dada una lista, cree una nueva lista con sólo los elementos que estaban en posición par. Utiliza `ennumerate` para identificar los índices pares. NM (**0.25 puntos**)

Nota: Considera que el primer elemento de la lista (en este caso, cat), tiene el índice 1.

[19]:
```
# Creamos la lista
list_animals = ['gato','perro','lobo','pez','hamster','leon','tigre','koala']
```

[20]:
```
## Respuesta
```

c) Crea un código que utilice las 3 listas para construir una frase con la siguiente estructura:

`El "animal" que pesa "weight" se puede tener en casa? "domestico"`

Recorre todos los elementos de las listas, de modo que resulte en 8 frases diferentes. La 1a frase sería:

`El gato que pesa 5kg se puede tener en casa? True`

Encuentra dos formas diferentes para obtener el mismo resultado. Una de ellas tiene que utilizar zip(). Cuál te parece más adecuada? Razona la respuesta. NM (**0.5 puntos**)

[21]:
```
# Creamos tres listas
list_animals = ['gato','perro','lobo','pez','hamster','leon','tigre','koala']
list_weight = [5, 12, 30, 0.5, 1, 50, 60, 15]
list_domestic = [True, True, False, True, True, False, False, False]
```

[22]:
```
## Respuesta
```

1.2.6 Ejercicio 6

Hemos encontrado un dataset con información de la población mundial. La primera fila contiene la cabecera, que define cada una de las columnas (Rank, Country, Capital, Continent, 2022_Population y Area). **(1 punto)**

a) Abre el archivo y muéstralo por pantalla. NM (**0.25 puntos**)

b) A continuación, responde a las siguientes preguntas:NM **(0.25 puntos)**

- Cuántas ciudades europeas hay?
- Cuál es el porcentaje de ciudades europeas respecto al total? Muestra el resultado con 3 decimales.

c) Por último, nos interesa tener la información de los países que comienzan por la letra `Z`. Crea un archivo nuevo sólo con las líneas de los países que empiezan por `Z`. NM (**0.5 puntos**)

d) 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**)

[23]:
```
## Respuesta
```