prog_datasci_2_python_ejerciciosResueltos

September 26, 2021

1 Fundamentos de Programación

1.1 Unidad 2: Breve introducción a la programación en Python 3 - Ejercicios resueltos

En este Notebook encontraréis un conjunto de ejercicios para practicar. Estos ejercicios no puntúan para la PEC, pero os recomendamos que los intentís resolver como parte del proceso de aprendizaje. Encontraréis ejemplos de posibles soluciones a los ejercicios en el 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 las actividades para practicar en el foro del aula.

1.2 Preguntas y ejercicios para practicar

1.2.1 Pregunta 1

¿Cuál es el valor final de a, b y c? (atención a los tipos de número decimal y entero)

a = 5 * 11

b = a / 2.

c, a = b * 2, b + 1

Respuesta

1.2.2 Pregunta 2

Identifica tres tipos de datos de Python y explica brevemente en qué se diferencian. Explica la diferencia entre el tipo lista y el tipo tupla.

Respuesta

1.2.3 Pregunta 3

¿Cuál es el resultado de la operación lógica `'manzana'> 'casa'`? Explica por qué.

Nota: Consultad en estándar Unicode para la codificación de caracteres.

Respuesta

1.2.4 Ejercicio 1

Escribir un programa que seleccione la cadena "analista" a partir de la cadena: "Ana es buena analista de datos.". Mostrar el resultado por pantalla.

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

1.2.5 Ejercicio 2

Qué expresión en Python necesitamos para conseguir el *string* "nohtyPythonython" utilizando sólo la palabra "Python"?

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

1.2.6 Ejercicio 3

Escribe un programa que asigne dos valores enteros cualquiera (elije un número entero aleatorio) a dos variables con nombre `a` y `b`. Utiliza las variables definidas anteriormente para evaluar la siguiente expresión matemática:

(a2 + b2)2

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

1.2.7 Ejercicio 4

Escribir un programa que calcule el volumen de una pirámide cuadrangular con una base de 4x5 metros de longitud y anchura respectivamente, y con una altura de 7,5 m. Recuerda que la fórmula para calcular el volumen es

donde l y w son la longitud y la anchura de la base, y h es la altura.

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

1.2.8 Ejercicio 5

Escribe un programa que defina una lista con el nombre de todos los meses del año. Haz que muestre los meses que corresponden a otoño.

[5]:
```
# Respuesta
```

1.2.9 Ejercicio 6

Escribe un programa que a partir de la lista de números dada muestre por pantalla una cadena de caracteres con todos los números, separados entre sí por un guión, y duplicando el primero y el último elemento de la lista.

Así, por ejemplo, para la lista ['1', '2', '3'] , el programa debería mostrar la cadena: '1-1-2-3-3'.

[6]:
```
# Respuesta
```

1.2.10 Ejercicio 7

Ordena la siguiente lista de cadenas de caracteres:

1. Invirtiendo el orden original
2. En orden alfabético inverso

Nota: Puedes consultar la documentación oficial de la función `sorted ()` para ver qué parámetros puedes utilizar para resolver la segunda parte de la actividad.

[7]:
```
st_chars = ["Benjamin Sisko", "Kira Nerys", "Odo", "Quark", "Jadzia Dax"]
```

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

1.2.11 Ejercicio 8

A partir de la siguiente lista,

`A_list = [42, 7.5, "Answer to the Ultimate Question", "Dave", 7.5]`

proporciona expresiones que retornen:

1. El número de veces que aparece el elemento `7.5` en la lista.
2. La posición de la primera aparición del valor `7.5`.
3. La misma lista sin el último elemento.

Nota: En el notebook de teoría hemos visto que son las listas y algunas operaciones sobre ellas. Para hacer la actividad, necesitaréis investigar algunas operaciones adicionales que podemos realizar sobre listas. Para ello podéis consultar la documentación oficial de Python sobre listas (intro y más sobre listas).

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

1.2.12 Ejercicio 9

¿Qué expresión en Python necesitamos para conseguir la *string* `Learning Python` utilizando sólo las variables`str1` y `str2` definidas?

[10]:
```
str1 = "I love Python, it's great!"
str2 = "Learning"

# Respuesta
```

1.2.13 Ejercicio 10

El codigo que se presenta a continuación no funciona:

Corrige el error y explica por qué sucede.

[11]:
```
name = "Alex"

age = 20
```

```
print("Hello, my name is %d and I'm %d years old!" % (name, age))
```

```
      ␣
↪---------------------------------------------------------------------------

       TypeError                                 Traceback (most recent call␣
↪last)

       <ipython-input-11-734b34da80ae> in <module>
         3 age = 20
         4
   ----> 5 print("Hello, my name is %d and I'm %d years old!" % (name, age))


       TypeError: %d format: a number is required, not str
```

1.2.14 Ejercicio 11

En este ejercicio utilizaremos la frase: La paciencia es la madre de la ciencia.

a) Extrae las palabras paciencia y ciencia de la cadena de strings definida anteriormente.

b) Crea la palabra ciencia a partir de paciencia, y la palabra paciencia a partir de ciencia.

c) Separa la frase en las diferentes palabras que la forman (e.g. “La”, “paciencia,” es ”, etc). ¿Cuál es el separador que debemos usar?

d) Vuelve a unir los elementos de la frase mediante “-” (e.g. “La-paciencia-es …”).

1.3 Soluciones a los ejercicio para practicar

1.3.1 Pregunta 1

¿Cuál es el valor final de a, b y c (atención a los tipos de número decimal y entero)?

a = 5 * 11

b = a / 2.

c, a = b * 2, b + 1

Iremos por pasos:

En la primera línea encontramos que la variable `a`, que contendrá el valor de 5* 11 (el primero un número entero y el segundo también un número entero, por lo que el resultado será entero), **55**. La variable `b`, al contrario, tendrá el valor de 55 (entero) dividido por 2. (decimal), por lo tanto, **27.5** (decimal).

En la tercera línea tenemos una expresión donde se asignan dos variables al mismo tiempo (en realidad esto es una asignación entre tuplas dado que ',' es el constructor de tuplas en Python). Primero evaluamos la parte de la derecha:

- `b * 2`, el resultado es decimal dado que también lo era antes: 27.5 * 2 = 55.0
- `b + 1`, el resultado es también decimal, 27.5 + 1 = 28.5

Ahora asignamos los resultados en la parte de la izquierda de la expresión:

- `c = b * 2` (la expresión que ya se había evaluado) = 55.0
- 'a = b + 1' = 28.5

Por lo tanto, tenemos:

a = 28.5, b = 27.5, c = 55.0

Ejecutamos el código en Python para ver que efectivamente este es el resultado:

[ ]:
```
a = 5 * 11
b = a / 2.
c, a = b * 2, b + 1

print (a, b, c)
```

1.3.2 Pregunta 2

Identifica tres tipos de datos de Python y explica brevemente en qué se diferencian. Explica la diferencia entre el tipo lista y el tipo tupla.

Respuesta

Integers: Hace referencia a valores numéricos enteros. En Python 3 no hay ningún límite en cuanto al tamaño de un valor entero. Por supuesto, está limitado por la cantidad de memoria que tiene el sistema.

Floats (Floating-Point Numbers): A diferencia del tipo *integer* o entero, el tipo *float* de Python designa valores flotantes o decimales que se especifican con un punto.

Strings: Las *strings* (o cadenas) son secuencias de datos de caracteres. El tipo cadena en Python se llama `str`. Los literales de cadena se pueden delimitar mediante comillas simples o dobles. Una cadena en Python puede contener tantos caracteres como se desean, el único límite son los recursos de memoria de la máquina.

En Python, las **listes** y las **tuplas** se declaran de diferentes maneras. Se crea una lista mediante corchetes `[]`, mientras que la tupla se declara con paréntesis `()`. Hay diferencias notables entre los dos, siendo la principal diferencia que las listas son mutables mientras que los tuplas son inmutables. Una lista tiene un tamaño variable mientras que una tupla tiene un tamaño fijo. Además, las operaciones sobre tuplas se pueden ejecutar más rápidamente en comparación con las de las listas.

1.3.3 Pregunta 3

¿Cuál es el resultado de la operación lógica `'manzana'> 'casa'`? Explica por qué.

Nota: Consultad en estándar Unicode para la codificación de caracteres.

Respuesta

El resultado de la operación es *True*. La comparación de cadenas Python se realiza mediante la evaluación de los caracteres de las dos cadenas, uno por uno. Cuando se encuentran diferentes caracteres, se compara su valor Unicode. El carácter con un valor Unicode inferior se considera más pequeño. En este caso "c" es menor si se compara con "m" a causa de sus valores Unicode correspondientes. A continuación utilizamos la función ord () para mostrar el valor del código Unicode de cada uno de los caracteres:

[ ]:
```
print('manzana' > 'casa')
print('El valor unicode de m es', ord('m'), 'y el de c es ', ord('c'))
```

1.3.4 Ejercicio 1

Escribir un programa que seleccione la cadena "analista" a partir de la cadena: "Ana es buena analista de datos.". Mostrar el resultado por pantalla.

[ ]:
```
# Respuesta
frase = "Ana es buena analista de datos."

palabra = frase[12:21]

print(palabra)
```

1.3.5 Ejercicio 2

Qué expresión en Python necesitamos para conseguir el *string* "nohtyPythonython" utilizando sólo la palabra "Python"?

Como podemos ver, para construir la cadena de texto necesitamos dos bloques diferenciados:

- la sub-cadena _ "nothyP" _ que corresponde a la palabra Python en orden inverso
- la sub-cadena _ "ython" _ repetida 2 veces

Podemos crear estos bloques de forma separada y al final concatenarlos para obtener el resultado final.

[ ]:
```
# Respuesta

# Primero definimos nuestra cadena base
python = "Python"

# Accedemos a los caracteres de Python en orden inverso
bloque1 = python[::-1]

# Accedemos a los 5 caracteres finales y los repetimos 2 veces
bloque2 = 2* python[1:]

# Finalmente, concatenar los bloques anteriores
resultado = bloque1 + bloque2
```

```
print (resultado)
```

1.3.6 Ejercicio 3

Escribe un programa que asigne dos valores enteros cualquiera (elije un número entero aleatorio) a dos variables con nombre `a` y `b`. Utiliza las variables definidas anteriormente para evaluar la siguiente expresión matemática:

(a2 + b2)2

[ ]:
```
# Elegimos dos números enteros al azar
a = 6
b = 5

# Calculamos los cuadrados de las variables anteriores utilizando el operador **
cuadrado_a = a ** 2
cuadrado_b = b ** 2

# Calculamos la suma de los cuadrados
suma_cuadrados = cuadrado_a + cuadrado_b

# Finalmente, calculamos el cuadrado de la suma
resultado = suma_cuadrados ** 2
print(resultado)

# También podemos escribir esta misma expresión en una sóla línea
resultado = (a**2 + b**2)**2
print(resultado)
```

1.3.7 Ejercicio 4

Escribir un programa que calcule el volumen de una pirámide cuadrangular con una base de 4x5 metros de longitud y anchura respectivamente, y con una altura de 7,5 m. Recuerda que la fórmula para calcular el volumen es

donde l y w son la longitud y la anchura de la base, y h es la altura.

[ ]:
```
# Respuesta

# Asignamos los valores del radio y la altura
base_long = 4
base_anchura = 5
altura = 7.5

# A continuación calculamos el volumen utilizando la siguente fórmula:
volumen = 1/3 * base_long * base_anchura* altura

print('El volumen de la pirámide es de ', volumen, 'metros cúbicos.')
```

1.3.8 Ejercicio 5

Escribe un programa que defina una lista con el nombre de todos los meses del año. Haz que muestre los meses que corresponden a otoño.

[ ]:
```
# Definimos la lista con los meses del año
meses = ["Enero", "Febrero", "Marzo", "Abril", "Mayo", "Junio", "Julio",␣
 ↪"Agosto", "Septiembre", "Octubre", "Noviembre", "Diciembre"]

# Los meses correspondientes a otoño son los meses 9, 10 y 11
print (meses[8:11])
```

Fijaros que en Python el primer elemento de una lista o array lo encontraremos en la posición **0**. Por lo tanto, si queremos acceder al noveno elemento de la lista utilizaremos el índice 8, es decir, escribiremos `meses [8]`.

Por otra parte, cuando especificamos un rango, `8: 11`, el elemento final no se incluye en el resultado final. Es decir, `meses [08:11]` devolverá los elementos correspondientes a las posiciones 8, 9 y 10.

1.3.9 Ejercicio 6

Escribe un programa que a partir de la lista de números dada muestre por pantalla una cadena de caracteres con todos los números, separados entre sí por un guión, y duplicando el primero y el último elemento de la lista.

Así, por ejemplo, para la lista ['1', '2', '3'] , el programa debería mostrar la cadena: '1-1-2-3-3'.

[ ]:
```
# Definimos la lista de números de entrada
numeros = ['6', '3', '45', '23', '2', '3', '17', '80']

# Respuesta
numeros = [numeros[0]] + numeros + [numeros[-1]]
print('-'.join(numeros))
```

1.3.10 Ejercicio 7

Ordena la siguiente lista de cadenas de caracteres:

1. Invirtiendo el orden original
2. En orden alfabético inverso

Nota: Podéis consultar la documentación oficial de la función [`sorted ()`] (https://docs.python.org/3/library/functions.html#sorted) para ver qué parámetros puedes utilizar para resolver la segunda parte de la actividad.

[ ]:
```
st_chars = ["Benjamin Sisko", "Kira Nerys", "Odo", "Quark", "Jadzia Dax"]

# Hemos visto el uso de los dos puntos (":") para acceder a múltiples elementos␣
 ↪de una lista.
# Python trata la expresión de la siguiente manera:
```

```
# - Si no hay ningún índice antes del primer ":" asume que debe comenzar por el␣
 ↪principio de la lista.
# - Si no hay ningún índice tras el primer ":" asume que debe posicionarse al␣
 ↪final de la lista.
# - El último número nos indica los incrementos en el índice que se deben hacer␣
 ↪a medida que recorremos la lista.

# (En este caso siempre "sumamos" -1 al índice y por lo tanto, obtenemos -1,␣
 ↪-2, -3, etc.)
print(st_chars[::-1])
```

Para el siguiente ejercicio, buscamos la documentación de la función `sorted ()` y vemos que esta nos sirve para ordenar cualquier secuencia (lista, tupla) y devuelverá una lista con los elementos de manera ordenada, sin modificar la secuencia original. El parámetro *reverse* predeterminado se establece como *falso*, pero si lo establecemos como *True*, entonces el Iterable ordenaría en orden inverso (descendente):

[ ]:
```
print(sorted(st_chars, reverse=True))
```

1.3.11 Ejercicio 8

A partir de la siguiente lista,

`A_list = [42, 7.5," Answer to the Ultimate Question "," Dave ", 7.5]`

proporciona expresiones que retornen:

1. El número de veces que aparece el elemento `7.5` en la lista.
2. La posición de la primera aparición del valor `7.5`.
3. La misma lista sin el último elemento.

Nota: En el notebook de teoría hemos visto que son las listas y algunas operaciones sobre ellas. Para hacer la actividad, necesitaréis investigar algunas operaciones adicionales que podemos realizar sobre listas. Para ello podéis consultar la documentación oficial de Python sobre listas (intro y más sobre listas).

[ ]:
```
a_list = [42, 7.5, "Answer to the Ultimate Question", "Dave", 7.5]

# El método count () cuenta cuántas veces aparece un valor en una lista y lo␣
 ↪devuelve.
print (a_list.count (7.5))

# Utilizaremos el método index (), éste encuentra el valor dado en una lista y␣
 ↪devuelve la posición del elemento.
# Si el mismo valor aparece más de una vez, el método index() devuelve la␣
 ↪posición de su primera aparición.
print (a_list.index (7.5))
```

```
# Indicamos que queremos excluir el último elemento indexado de la lista usando␣
 ↪el operador []:
print (a_list [: - 1])
```

1.3.12 Ejercicio 9

¿Qué expresión en Python necesitamos para conseguir la *string* `Learning Python` utilizando sólo las variables`str1` y `str2` definidas?

[ ]:
```
str1 = "I love Python, it's great!"
str2 = "Learning"
```

Como podemos ver, para construir la cadena de texto necesitamos aislar la sub-cadena "Python", que encontramos en la tercera palabra de la cadena `str1` y corresponde con los caracteres 7 a 13 de la cadena. Podremos crear esta nueva cadena de forma separada y luego concatenarla con la cadena `str2` utilizando el operador `+`:

[ ]:
```
# Accedemos a los caracteres que nos interesan de la cadena str1, incluyendo el␣
 ↪espacio previo a la palabra 'Python'
subcadena = str1 [6:13]

# Finalmente, concatenamos la nueva cadena (subcadena 'Python') y la␣
 ↪concatenamos con str2
resultat = str2 + subcadena

# Mostramos el resultado
print (resultat)
```

[ ]:
```
# También podemos añadir el espacio en blanco entre palabras al unirlas,␣
 ↪cuidado con no añadir dos espacios en blanco:
subcadena = str1 [7:13]
resultado = str2 + ' ' + subcadena
print (resultado)
```

1.3.13 Ejercicio 10

El codigo que se presenta a continuación no funciona:

Corrige el error y explica por qué sucede.

[ ]:
```
name = "Alex"

age = 20

print("Hello, my name is %d and I'm %d years old!" % (name, age))
```

[ ]:
```
# Respuesta

```

```
name = "Alex"

age = 20

print("Hello, my name is %s and I'm %d years old!" % (name, age))
```

Respuesta

Para poder utilizar esta funcionalidad, debe haber una concordancia entre el tipo de la variable que queremos sustituir y la expresión utilizada. Es decir, si queremos sustituir un string, debemos utilizar la expresión `%s` que indica que el elemento que queremos referenciar es también un string. Si utilizamos `%d` y el contenido es string, Python devolverá un mensaje de error, ya que hay una discrepancia en cuanto al tipo de variable.

1.3.14 Ejercicio 11

En este ejercicio utilizaremos la frase: La paciencia es la madre de la ciencia.

a) Extrae las palabras paciencia y ciencia de la cadena de strings definida anteriormente.

b) Crea la palabra ciencia a partir de paciencia, y la palabra paciencia a partir de ciencia.

c) Separa la frase en las diferentes palabras que la forman (e.g. “La”, “paciencia,” es ”, etc). ¿Cuál es el separador que debemos usar?

d) Vuelve a unir los elementos de la frase mediante “-” (e.g. “La-paciencia-es …”).

[ ]:
```
# Respuesta

# Creamos la frase del enunciado
frase = "La paciencia es la madre de la ciencia"

# a)

# Seleccionamos los elementos necesarios para obtener las palabras paciencia y␣
↪ciencia
palabra1 = frase[3:13]
palabra2 = frase[-7:]

# Mostramos el resultado por pantalla
print("La primera palabra es:",palabra1)
print("La segunda palabra es:",palabra2)
```

[ ]:
```
# b)

# Creamos la palabra ciencia a partir de paciencia
palabra3 = palabra1[2:]

# Creamos la palabra ciencia a partir de paciencia
palabra4 = 'pa'+ palabra2
```

```
# Mostramos el resultado por pantalla
print("La tercera palabra es:",palabra3)
print("La quarta palabra es:",palabra4)
```

[ ]:
```
# c)

# Separamos las palabras utilizando el separador " "
frase_mod = frase.split(' ')

# Mostramos el resultado por pantalla
print("La frase separada por sus elementos es:",frase_mod)
```

[ ]:
```
# d)

# Especificamos el separador
s = "-"

# Utilizamos la función join para unir los elementos de la lista
frase_joined = s.join(frase_mod)

# Mostramos el resultado por pantalla
print("La frase con el nuevo separador es:",frase_joined)
```