Trabajo Condicionales y Bucles 01

August 19, 2019

1 Descripción de la actividad

A continuación se enuncian los criterios de la actividad relacionada con *Condicionales* y *Bucles* en **Python**:

1. El resultado final ha de ser un notebook de *Jupyter* que contenga el enunciado y la solución de cada uno de los ejercicios asignados.
2. El código DEBE IR COMENTADO para facilitar su comprensión.
3. El código DEBE FUNCIONAR CORRECTAMENTE.
4. Todos los códigos deben ser en **Python 3**.
5. El grupo debe estar en capacidad de SUSTENTAR SUS CÓDIGOS si se llegase a considerar necesario por el docente. En este caso la justificación decide la calificación.

A cada uno de los grupos (de máximo 3 personas) se les asignará dos problemas.

> **IMPORTANTE:** El código presentado NO debe usar funciones nativas de Python ni de ninguna librería. Debe estrictamente usar los condicionales y bucles para cumplir su labor. Se excluyen de esta regla, por supuesto, los operadores (aritméticos, lógicos, relacionales) y las funciones `len()` y `print()`.

Cualquier duda o asesoría adicional no dudar en consultar con el docente de la asignatura. Adicionalmente, al final de este documento se encuentra una sección de *Ayudas* con elementos útiles en los cuales se puede ayudar para resolver mejor los ejercicios.

2 Ejercicios

2.1 Ejercicio 1

Crear un código al cual al entregársele una lista con las calificaciones de un estudiante (en un rango de 0.0 a 5.0), determine el promedio de notas y si el estudiante aprobó o no (nota aprobatoria de 3.0 o superior).

2.2 Ejercicio 2

Crear un código el cual permita determinar si un número entero positivo dado es primo o no.

> **NOTA:** Considere usar el bucle `for` junto con la cláusula `else`.

2.3 Ejercicio 3

Crear un código que permita determinar si un año dado es bisiesto o no.

> **NOTA:** Un año es bisiesto si es divisible entre 4, salvo los que son divisibles entre 100 que no sean a su vez divisibles entre 400 (Ej.: 2019 no es bisiesto, 2020 sí, 2100 no será bisiesto pero 2400 sí.)

2.4 Ejercicio 4

Crear un código que a partir de dos listas (una para las notas de un estudiante y otra para su peso porcentual dentro del promedio) permita determinar el promedio ponderado de notas.

2.5 Ejercicio 5

Crear un código que a partir de un primer término y la diferencia constante (distinta de cero), entregue una lista con los primeros 10 términos de una **progresión aritmética**, la suma de estos y diga si esta progresión es creciente o decreciente.

2.6 Ejercicio 6

Crear un código que a partir de un primer término (distinto de cero) y la razón constante (distinta de cero o de la unidad), entregue los primeros 10 términos de una **progresión geométrica**, la suma de estos y diga si esta progresión es creciente o decreciente.

2.7 Ejercicio 7

Crear un código que al indicársele el número al que corresponda un día del año indique (de 0 a 365) diga la fecha para ese día (mes y día). Asuma que no se trata de un año bisiesto.

2.8 Ejercicio 8

Crear un código que tras indicársele la estatura y el peso de una persona, indique si índice de masa corporal (BMI) y a qué categoría pertenece.

> **NOTA:** El BMI se calcula como el cociente entre el peso de la persona (en kg) y el cuadrado de su estatura (en metros). Las categorías básicas son las siguientes: *bajo peso* (BMI $<$ 18.5), *normal* (18.5 $\leq$ BMI $<$ 25), *sobrepeso* (25 $\leq$ BMI $<$ 30) y *obesidad* (BMI $\geq$ 30).

2.9 Ejercicio 9

Crear un código para el que dado un número entero positivo entregue una lista con todos sus divisores.

2.10 Ejercicio 10

Es posible lograr una buena aproximación del número π mediante la siguiente serie:

$$\pi \approx 4\sum_{i=0}^{n}\frac{(-1)^i}{2i-1} = 4\left(\frac{1}{1}-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\cdots+\frac{1}{2n-1}\right)$$

Cree un código que permita encontrar una aproximación de π para una cantidad dada de términos $(n-1)$ de esta suma.

> **NOTA:** Mientras mayor sea la cantidad de términos de la suma, mejor la aproximación.

2.11 Ejercicio 11

Escriba un código que permita generar una lista con los primeros n términos de la secuencia de Fibonacci. En esta sucesión los dos primeros términos son el 1, mientras que los siguientes se obtienen sumando los dos términos anteriores: $1, 1, 2, 3, 5, 8, 13, 21, \ldots$.

2.12 Ejercicio 12

Un objeto se lanza desde una altura inicial dada (y_0, en metros) y con una velocidad inicial dada (v_0, en m/s). Usando el bucle `while` cree un código que determine la altura máxima (aproximada) del cuerpo tras ser lanzado indicando además el instante final (aproximado a las milésimas de segundo) en que lo ha logrado.

> **NOTA:** Tome como consideración para determinar la altura máxima si la altura anterior es mayor que la actual. NO use otra fórmula de física que no sea $y = y_0 + v_0 t - \frac{1}{2}gt^2$ (con $g = 9.8$).

3 Ayudas

A continuación encontrará algunos tips que le ayudarán a solventar de manera más eficiente algunos de los ejercicios:

3.1 Determinar si un número es divisible o no por otro

El operador aritmético `%` permite obtener el resto de la división entera. Por ejemplo, al dividir 23 entre 7 el resultado es 3 y sobran 2. Obsérvese cómo se obtienen estos valores en Python:

[1]:
```
print(23 // 7) # Para la división entera (3).
print(23 % 7)  # Para el resto de la división entera (2).
```

```
3
2
```

De esta manera, si queremos saber si un número divide a otro basta con determinar si el resto de la división entera es cero o no. Veamos dos maneras posibles de hacerlo:

[2]:
```
# Determinemos si 35 es divisible entre 5:
# El operador de comparación == equivale a "ser igual a"
```

```
print(35 % 5 == 0) # True indica que sí es divisible; False indica que no.
```

```
True
```

[3]:
```
# El operador != equivale a "ser distinto de":

print(35 % 5 != 0) # False indica que sí es divisible; True que no lo es.
```

```
False
```

3.2 Crear una lista de manera automática

En algunos ejercicios se precisa entregar una lista con valores obtenidos tras operar un bucle. Esto se consigue fácilmente con el *método* para listas `append()`.

A continuación se muestra un ejemplo de su uso: crear una lista con los cuadrados de los primeros n enteros positivos:

[4]:
```
# Definimos la cantidad de enteros positivos a considerar:
n = 8

# Creamos una lista vacía
lista = []

# Creamos el bucle que agregará los elementos a la lista
for i in range(1, n+1):
    lista.append(i**2) # Añade al final de la lista cada i^2

print(lista) # Imprime la lista final
```

```
[1, 4, 9, 16, 25, 36, 49, 64]
```

3.3 Los conectores lógicos son muy útiles

El correcto uso de los conectores lógicos pueden simplificar enormemente un código. Los conectores básicos en Python son `&` (o también `and`), `|` (o también `or`), `^` y `not`; los cuales representan, respectivamente a "*y*" (conjunción), "*o*" (disyunción inclusiva), "*ó*" (disyunción exclusiva) y "*no*" (negación). Su uso es como en lógica matemática. Veamos:

[5]:
```
print(True & False) # Verdadero Y Falso (da Falso)
print(True | False) # Verdadero O Falso (da Verdadero)
print(True ^ False) # Verdadero Ó Falso (da Verdadero)
print(not(True))    # No Verdadero (da Falso)
```

```
False
True
True
False
```