

EJERCITACIÓN Nº 2: FUNCIONES Y DECISIONES

Cátedra Programación II


Marzo 2019

1. Expresiones Booleanas

Una expresión booleana es una expresión que puede ser verdadera (True) o falsa (False).

Las constantes True y False son valores especiales, los cuales pertenecen al tipo *bool*, no son *int*, ni *float*, tampoco son *strings*.

Los *Operadores Relacionales de Comparación* en Python se muestran en el siguiente recuadro, y los mismos tomando expresiones de distinto tipos, pero coinciden en la mayoría de sus casos entre los operandos, nos devuelven un resultado de tipo *booleano*.

 Operadores de Comparación	
Expresión	Significado
<code>a == b</code>	a es igual a b
<code>a != b</code>	a es distinto de b
<code>a < b</code>	a es menor que b
<code>a <= b</code>	a es menor o igual que b
<code>a > b</code>	a es mayor que b
<code>a >= b</code>	a es mayor o igual que b

EJERCICIO 1. Las siguientes expresiones utilizan distintos operadores relacionales los cuáles comparan dos operandos y determinan su validez. Ejecutar los mismos en el intérprete de Python y analizar los resultados.

```
1 >>> 5 == 5
2 >>> 5 > 6
3 >>> 5 <= 6
4 >>> 5.2 == 3.1
5 >>> 5.2 == 5.2
6 >>> 5.0 == 5
7 >>> 5.0 >= 5
8 >>> 5.0 <= 5
9 >>> True == True
10 >>> False == False
11 >>> "Casa" == "casa"
12 >>> "Casa" != "casa"
13 >>> "casa" == "casa"
14 >>> "casa" == 1
15 >>> "Casa" != 1
16 >>> "1" != 1
```

17


```
>>> "Vilma" > "Betty"
```

Código 1: Expresiones Booleanas

EJERCICIO 2. ¿Cuál de las siguientes expresiones son expresiones booleanas? Piense el resultado y luego verifique los mismos usando el interprete de python.

- (A) True
- (B) 3 == 4
- (C) 3 + 4
- (D) 3 + 4 == 7
- (E) "False"

Los *Operadores Lógicos* son los utilizados para componer, o negar distintas expresiones booleanas, generando así nuevas expresiones booleanas. En Python utilizamos los siguientes operadores: and, or, y not para representar a las operaciones lógicas conocidas como: *conjunción*, *disyunción inclusiva*, y la *negación*.

 Operadores de Lógicos	
Expre- sión	Significado
a and b	El resultado es True solamente si a es True y b es True de lo contrario el resultado es False
a or b	El resultado es True si a es True o b es True (o ambos) de lo contrario el resultado es False
not a	El resultado es True si a es False de lo contrario el resultado es False

En Python, como en la mayoría de los lenguajes de programación se utilizan una *booleana de cortocircuito*, la cual una vez que se evalúo parte de una expresión lógica compuesta, y se determinó mediante el primer operando el resultado de la expresión, los restantes operando nunca serán evaluados.

```

1
2 #Debe evaluar ambas subexpresiones, el primer operando es True
3 >>> x = 5
4 >>> print("Caso 1: ", x > 0 and x < 10)
5
6 #Sólo evalúa la primera expresión, por "and-cortocircuito" es falsa.
7 >>> x = -5
8 >>> print("Caso 2: ", x > 0 and x < 10)
9
10 #Debe evaluar ambas subexpresiones, el primer operando del or es False
11 >>> n = 25

```

```
12 >>> print("Caso 3: ", n % 2 == 0 or n % 3 == 0)
13
14 #Sólo evalúa al primer operando del or por ser este True
15 >>> n = 24
16 >>> print("Caso 4: ", n % 2 == 0 or n % 3 == 0)
```

Código 2: Expresiones Booleanas Evaluaciones Cortocircuito

EJERCICIO 3. Indique de las siguientes opciones, cuál se correcta, si lo que buscamos es chequear que el valor almacenado en una variable x esta comprendido entre 0 y 5 sin incluir ambos extremos.

- (A) $x > 0$ and < 5
- (B) $x > 0$ or $x < 5$
- (C) $x > 0$ and $x \leq 5$
- (D) $x \geq 0$ and $x \leq 5$

EJERCICIO 4. Indique de las siguientes opciones, cuál se correcta, si lo que buscamos es chequear que el valor almacenado en una variable x es igual a los valores 6, 7 u 8.

- (A) $x == 6$ or 7 or 8
- (B) $x == 6$ or $x == 7$ or $x == 8$
- (C) $x \geq 6$ and $x \leq 8$

Las *propiedades de asociatividad* entre los operadores nos permite determinar la precedencia entre los operadores cuando estos comparten el mismo nivel. La mayoría de los operadores en Python tienen asociatividad de izquierda a derecha (izq-der). Por ejemplo, la multiplicación y la división entera, tienen la misma precedencia, entonces, si ambos operadores están presentes en una expresión el que este más a la izquierda se evaluará primero. En otro caso, como sucede con el operador de exponente ($**$), este posee asociatividad de derecha a izquierda (der-izq). Entonces cuando tengamos más de un exponente evaluaremos primeros las potencias, y luego se la aplicaremos a la base para obtener el resultado. De querer expresar otro comportamiento, entonces debemos agregar paréntesis para forzar otra asociatividad. A continuación mostraremos algunos ejemplos para ilustrar estos hechos:

Ejemplos:

```
1 # Asociatividad izq-der
2 # ((5 * 2) // 3)
3 # Resp: 3
4 >>> print(5 * 2 // 3)
5
6 # Forzamos asociatividad der-izq
7 # Resp: 0
```

```

8 >>> print(5 * (2 // 3))
9
10 # Asociatividad izq-der
11 # ((True and False) or True)
12 # Resp: true
13
14 >>> True and False or True
15
16 # Asociatividad der-izq
17 # (2 ** (3 ** 2))
18 # Resp: 512
19 print(2 ** 3 ** 2)
20
21 # Forzamos asociatividad izq-der de **
22 # Resp: 64
23 print((2 ** 3) ** 2)

```

Código 3: Expresiones Booleanas Asociatividades

Algunos operadores como la asignación y los operadores de comparación *no son asociativos* en Python. En aquellos casos que escribamos una secuencia de expresiones de asignación ésta es interpretada como tres asignaciones por separado, igualadas todas ellas a un mismo valor, señalado al final de la expresión. Del mismo modo que: $x < y < z$, no significa: $(x < y) < z$ ni $x < (y < z)$; sino que $x < y < z$ es igual a $x < y$ and $y < z$, y es evaluado de izquierda a derecha. Lo explicaremos en el siguiente párrafo como encadenamiento de los operadores de comparación.

Ejemplos:

```

1 #No es asociativa la asignación,
2 #si se puede encadenar con reglas propias de interpretación.
3 >> x=z=y=1
4 >>> print ("x: ", x, "y: ", y, "z:", z)

```

Código 4: No asociatividad

EJERCICIO 5. Verificar las asociatividad de cada operador aritmético, utilizando el interprete de Python. A continuación de cada expresión hemos señalado la respuesta que esperamos de la evaluación de esa expresión. Complete con paréntesis alrededor de cada expresión la asociación de términos que se realizó.

Para la "+" y la "-"

- a) $3 + 4 + 5$ (ve: 12)
- b) $3 - 4 - 5$ (ve: -6)
- c) $3 + 4 - 5$ (ve: 2)
- d) $3 - 4 + 5$ (ve: 4)

Para la "*" y la "/"

- a) $3 * 4 * 5$ (ve: 60)
- b) $3 / 4 / 5$ (ve: 0.15)
- c) $3 * 4 / 5$ (ve: 2.4)
- d) $3 / 4 * 5$ (ve: 3.75)

Para la "**" (potencia)

- a) $2 ** 2 ** 3$ (ve: 256)
- b) $3 ** 2 ** 2$ (ve: 81)

Para todas las operaciones juntas:

- a) $2 + 3 * 4 - 5 / 6 ** 7$ (ve: 13.99)
- b) $2 ** 3 / 4 + 5 - 6 * 7$ (ve: -35.0)

Algunos operadores relaciones se pueden *encadenar* como se muestra a continuación:

```

1 >>> x = 2
2 >>> 1 < x < 3
3 # Reinterpreta: (1 < x) and (x < 3)
4 True
5
6 >>> 10 < x < 20
7 #Reinterpreta: (10 < x) and (x < 20)
8 False
9
10 >>> 3 > x <= 2
11 #Reinterpreta: (3 > x) and (x <= 2)
12 True
13
14 >>> 2 == x < 4
15 #Reinterpreta: (2== x) and (x < 4)
16 True

```

Código 5: Expresiones Booleanas Encadenadas

Las comparaciones se realizan entre cada par de términos que serán evaluados. Por ejemplo, en el primer caso : $1 < x < 3$, la expresión $1 < x$ es True y es evaluada primero, y luego se evalúa la expresión $x < 3$, la cuál también es True, y por ello nos devuelve True. Un término por vez, y no los dos juntos, mas allá que los escribamos encadenadamente. La *expresión se reinterpreta* antes de evaluarla, el encadenamiento no es otra cosa la *propiedad de conjuntividad*, de ciertos operadores relaciones. Los ejemplos muestran el procedimiento.

2. Decisiones: if-else-elif



Decisión Simple

```
1 if <condición>:  
2     # acciones a ejecutar si condición es verdadera
```

- Bloque condicional simple. Las acciones a ejecutar se ejecutarán si la condición es verdadera.
- Si la condición es falsa, el programa continua su ejecución con las instrucciones subsiguientes al if.
- La sangría de las instrucciones es importante para determinar cuál es el bloque de código que se ejecutará bajo la condición impuesta.



Decisión con else

```
1 if <condición>:  
2     # acciones a ejecutar si condición es verdadera  
3 else:  
4     # acciones a ejecutar si condición es falsa
```

- La **condición** siempre es una expresión booleana.
- Bloque condicional con sentencia else. Las acciones que siguen al if se ejecutarán si la condición es verdadera, y las acciones que siguen al else se ejecutarán si la condición es falsa.
- Sólo se puede utilizar else si hay un if correspondiente.
- La sentencia else debe escribirse al mismo nivel que if, y las acciones a ejecutar deben tener un nivel de sangría mayor.

**Alternativas Encadenadas**

```
1
2 if <condición1>:
3     # acciones a ejecutar si condición1 es verdadera
4 elif <condición2>:
5     # acciones a ejecutar si condición2 es verdadera
6 elif <condición3>:
7     # acciones a ejecutar si condición3 es verdadera
8 else:
9     # acciones a ejecutar si ninguna de las condiciones anteriores fue↔
    verdadera.
```

- Cada bloque que se ejecuta si no se cumplieron las condiciones anteriores, pero sí se cumple la condición especificada.
- Sólo se puede utilizar `elif` si hay un `if` correspondiente, se lo debe escribir al mismo nivel que `if`, y las acciones a ejecutar deben escribirse en un bloque de sangría mayor.
- Puede haber tantos `elif` como se quiera, todos al mismo nivel.

2.1. Sólo ifs

EJERCICIO 6. Diseñar tres funciones que resuelvan los siguientes problemas:

- Dado un número entero n , indicar si es par o no.
- Dado un número entero n , indicar si es múltiplo de tres o no.
- Dado un número entero n , y un factor f indicar si es múltiplo de f o no.

EJERCICIO 7. Diseñar una implementación propia de la función `abs`, que devuelva el valor absoluto de cualquier valor que reciba.

EJERCICIO 8. Diseñar un programa, que utilizando funciones matemáticas, calcule las raíces reales (no las complejas) de un polinomio de segundo grado.

Nota: validar que las operaciones puedan efectuarse antes de realizarlas (no dividir por cero, ni calcular la raíz de un número negativo). La función no tienen que devolver nada, simplemente puede mostrar por pantalla el resultado del cálculo.

EJERCICIO 9. Diseñar funciones que resuelvan los siguientes problemas:

- Dado un año indicar si es bisiesto.
Nota: un año es bisiesto si es un número divisible por 4, pero no si es divisible por 100, excepto que también sea divisible por 400.
- Dado un mes, devolver la cantidad de días correspondientes.
- Dado un mes, representado como un número, devolver el mes asociado en palabras. Así para el valor 1 la función devolverá, “Enero”, y para 4 devolverá “Abril”, etc.

- d) Traducir una fecha dado su mes en formato numérico, a una fecha con su mes escrito en palabras. Por ejemplo, para `fecha_palabra(1,3,2019)` una traducción posible es "1 de Marzo de 2019".

Nota: en todos los casos, invocar las funciones escritas previamente cuando sea posible, y validar los datos de entrada.

EJERCICIO 10. Diseñar dos funciones que permitan calcular:

- La duración en segundos de un intervalo dado en horas, minutos, y segundos.
- La duración en horas, minutos y segundos de un intervalo dado en segundos.

Nota: en todos los casos validar los datos de entrada, los *segundos* los *minutos* deben estar comprendidos entre 0 y 59 inclusive, para las *horas* no pondremos restricciones.

EJERCICIO 11. Usando las funciones del ejercicio anterior, escribir un programa que pida al usuario dos intervalos expresados en horas, minutos y segundos, sume sus duraciones, y muestre por pantalla la duración total en horas, minutos, y segundos.

EJERCICIO 12. Definir la función `observar_clima` la cual clasifica con una "etiqueta" ciertos intervalos numéricos asociados a temperaturas ambientes. La categorización de las temperaturas es la siguiente:

Intervalo	Denominación
menos de 0 grados	"Helado"
entre 0 y 15 grados	"Frío"
entre 15 y 25 grados	"Agradable"
más de 25 grados	"Caluroso"

EJERCICIO 13. Modificar el programa anterior para satisfacer los siguientes requerimientos:

- Se diferencia una categoría más, ahora si la temperatura esta entre 25 y 32 grados el mensaje deberá ser "Caluroso", pero si la temperatura excede los 32 grados deberá decir "Muy caluroso".
- Se establece un nuevo criterio de clasificación:

Intervalo	Denominación
menos de 0 grados	"Helado"
entre 0 y 10 grados	"Frío"
entre 10 y 15 grados	"Fresco"
entre 15 y 25 grados	"Agradable"
entre 25 y 32 grados	"Caluroso"
más de 32 grados	"Muy Caluroso"

Modificar el programa para que se adapte a este cambio.

- El programa ahora deberá avisarnos si es necesario utilizar protector solar o no. Para ello, se sabe que, si la temperatura es menor a los 15 no es necesario utilizar protector solar, pero por el contrario si la temperatura supera dicho valor, si se debe utilizar protector.

- d) ¿Todas las modificaciones que hicimos sobre nuestro programa fueron iguales? Existe alguna diferencia entre ellas. Explicar.

EJERCICIO 14. Suponiendo que el primer día del año fue lunes, escribir una función que reciba un número con el día del año (de 1 a 366) y devuelva el día de la semana que le toca. Por ejemplo: si recibe '3' debe devolver 'miércoles', si recibe '9' debe devolver 'martes'.

EJERCICIO 15. Programa de astrología: el usuario debe ingresar el día y mes de su cumpleaños y el programa le debe decir a qué signo corresponde.

Aries: 21 de marzo al 20 de abril.	Tauro: 21 de abril al 20 de mayo.
Geminis: 21 de mayo al 21 de junio.	Cancer: 22 de junio al 23 de julio.
Leo: 24 de julio al 23 de agosto.	Virgo: 24 de agosto al 23 de septiembre.
Libra: 24 de septiembre al 22 de octubre.	Escorpio: 23 de octubre al 22 de noviembre.
Sagitario: 23 de noviembre al 21 de diciembre.	Capricornio: 22 de diciembre al 20 de enero.
Acuario: 21 de enero al 19 de febrero.	Piscis: 20 de febrero al 20 de marzo.

EJERCICIO 16. Escribir un programa que reciba como entrada un entero representando un año (por ejemplo 751, 1999, o 2158), y muestre por pantalla el mismo año escrito en números romanos.

2.2. Revisitando problemas de Racket

Reprogramar cada uno de los siguientes ítems. Estos problemas ya fueron resueltos el cuatrimestre pasado utilizando el lenguaje *Racket*. Repensarlos en esta oportunidad utilizando el lenguaje *Python*, reutilizando la receta de los mismos. ¿Cuánto nos sirve de lo ya hecho?

EJERCICIO 17.

- Defina la función `vol-cubo` que recibe la longitud de la arista de un cubo y calcula su volumen.
- Defina la función `area-cubo` que recibe la longitud de la arista de un cubo y calcula su área.
- Defina una función que según el valor que ingrese por el usuario, se calcule el volumen, o el área de un cubo. Tener en consideración un valor por defecto, si el valor ingresado no fuera el adecuado, en dicho caso deberá calcular el área de un cubo. Reutilizar las funciones de los ítems anteriores.
- Modifique la función definida en ítem anterior para que, en caso de no recibir un valor apropiado nos muestre el siguiente mensaje en pantalla *"Valor no soportada por la función."*

Nota: en todos los casos validar los datos de entrada, por ejemplo, es un error que la arista de un cubo sea 0 o un valor negativo, en dichos casos puede considerar retornar un mensaje de error, como por ejemplo, *"Valores de Entrada Inválidos"* o para no modificar el tipo de dato de retorno de cada función, para que contenga además `String`, retorna un número negativo, el cual es

inválido como resultado válido, y la funciones futuras que utilizan a ésta puede determinar así que se trata de un error.

2.3. Ciclos definidos for con ifs

EJERCICIO 18. Diseñe un programa que imprima los primeros 25 números naturales pares.

EJERCICIO 19. Diseñe un programa que imprima los primeros 100 números naturales pares.

1. ¿Pudo usar la solución propuesta en el ejercicio anterior?
2. ¿Puede pensar una solución que sirva para mostrar los primeros n números naturales pares?

EJERCICIO 20. Diseñe un programa que imprima los primeros n números pares mayores que m . ¿Puede adaptar la solución propuesta en el ejercicio anterior para que resuelva este problema?

EJERCICIO 21. Diseñe un programa que permita clasificar 23 notas de un examen de distintos alumnos ingresadas por un auxiliar docente. Para ello, el docente estableció el siguiente criterio de aprobación, el cual el programa deberá imprimir, luego de cargada la nota.

Nota	Condición
$1 \leq nota < 4$	<i>Insuficiente</i>
$4 \leq nota < 6$	<i>No Aprobado</i>
$6 \leq nota < 8$	<i>Aprobado</i>
$8 \leq nota < 10$	<i>Muy Bueno</i>
$nota = 10$	<i>Excelente</i>

EJERCICIO 22.

- a) Teniendo en cuenta lo realizado en el programa anterior, diseñar ahora una programa clasificar 10 notas, donde ahora la nota se obtendrá a partir del promedio de las tres notas obtenidas por el estudiante en el cuatrimestre. Así, el docente ingresará las tres notas del alumno 1, y el programa clasificará el promedio (previo cálculo) utilizando el criterio explicado en el ejercicio anterior. Luego, ingresará la notas del alumno 2, y clasificará el promedio, y así sucesivamente, hasta completar con la clasificación de los 10 alumnos
- b) ¿Qué modificaciones realizaría en su programa si el docente ahora toma 4 parciales durante el cuatrimestre? ¿Y si ingresa 7 notas por alumno?
- c) ¿Qué modificaciones realizaría en su programa si el docente ahora recibe 15 alumnos? ¿Y si debe las notas de 50 alumnos?

Referencias

[1] Think Python: How to Think Like a Computer Scientist, Allen B. Downey, 2nd Edition, Version 2.2.18.

- [2] Algoritmos y Programación I, Aprendiendo a programar usando Phyton como herramienta, Rosita Wachenchauzer et.al., 2016, (sin publicar).