Módulo 3 - Valores booleanos, ejecución condicional, bucles, listas y su procesamiento, operaciones lógicas y de bit a bit

Tomando decisiones

- = asignación
- == comparación, es igual
- != no es igual
- >,<,>=,⇐

if

```
if <exp>:
    linea1
    linea2
  elif <exp>:
    linea5
    linea6
  else:
    linea3
    linea4
```

```
if number1 > number2: larger_number = number1
else: larger_number = number2
```

- no mezclar tabs y espacios en blanco en la indentación
- `else` es la última rama de la cascada, opcionalmente

bucles (ciclos)

- algoritmo, pseudocódigo
- `max()`: máximo de X números → `min()`

ejercicio

```
Érase una vez una tierra de leche y miel, habitada por gente feliz y próspera. La
gente pagaba impuestos, por supuesto, su felicidad tenía límites. El impuesto más
importante, denominado Impuesto Personal de Ingresos (IPI, para abreviar) tenía que
pagarse una vez al año y se evaluó utilizando la siguiente regla:

Si el ingreso del ciudadano no era superior a 85,528 pesos, el impuesto era igual
al 18% del ingreso menos 556 pesos y 2 centavos (esta fue la llamada exención
fiscal ).
Si el ingreso era superior a esta cantidad, el impuesto era igual a 14,839 pesos y
2 centavos, más el 32% del excedente sobre 85,528 pesos.
Tu tarea es escribir una calculadora de impuestos.
```

```
Debe aceptar un valor de punto flotante: el ingreso.
A continuación, debe imprimir el impuesto calculado, redondeado a pesos totales.
Hay una función llamada round() que hará el redondeo por ti, la encontrarás en el
código de esqueleto del editor.
Nota: Este país feliz nunca devuelve dinero a sus ciudadanos. Si el impuesto
calculado es menor que cero, solo significa que no hay impuesto (el impuesto es
igual a cero). Ten esto en cuenta durante tus cálculos.

Observa el código en el editor: solo lee un valor de entrada y genera un resultado,
por lo que debes completarlo con algunos cálculos inteligentes.
```

```
income = float(input("Introduce el ingreso anual:"))

if income<=85528:
    tax = (income * 18 / 100) - 556.2
else:
    tax = 14839.2 + (income - 85528) * 32 / 100

if tax < 0: tax = 0.0

tax = round(tax, 0)
print("El impuesto es:", tax, "pesos")
```

ejercicio

```
Como seguramente sabrás, debido a algunas razones astronómicas, el año pueden ser
bisiesto o común. Los primeros tienen una duración de 366 días, mientras que los
últimos tienen una duración de 365 días.

Desde la introducción del calendario Gregoriano (en 1582), se utiliza la siguiente
regla para determinar el tipo de año:

Si el número del año no es divisible entre cuatro, es un año común.
De lo contrario, si el número del año no es divisible entre 100, es un año
bisiesto.
De lo contrario, si el número del año no es divisible entre 400, es un año común.
De lo contrario, es un año bisiesto.
Observa el código en el editor: solo lee un número de año y debe completarse con
las instrucciones que implementan la prueba que acabamos de describir.

El código debe mostrar uno de los dos mensajes posibles, que son Año Bisiesto o Año
Común, según el valor ingresado.

Sería bueno verificar si el año ingresado cae en la era Gregoriana y emitir una
advertencia de lo contrario: No dentro del período del calendario Gregoriano.
Consejo: utiliza los operadores != y %.
```

```
year = int(input("Introduce un año:"))

if year >= 1582:
    if year % 4 != 0: ano="común"
    elif year % 100 != 0: ano="bisiesto"
    elif year % 400 != 0: ano="comun"
```

```
    else: ano="bisiesto"
    print("Año "+ano)
else:
    print("no es gregoriano")
```

while

```
while conditional_expression:
    instruction
```

```
while True:
    print("Estoy atrapado dentro de un bucle.")
```

- Si deseas ejecutar más de una sentencia dentro de un while, debes (como con if) poner sangría a todas las instrucciones de la misma manera.
- Una instrucción o conjunto de instrucciones ejecutadas dentro del while se llama el cuerpo del bucle.
- Si la condición es False (igual a cero) tan pronto como se compruebe por primera vez, el cuerpo no se ejecuta ni una sola vez (ten en cuenta la analogía de no tener que hacer nada si no hay nada que hacer).
- El cuerpo debe poder cambiar el valor de la condición, porque si la condición es True al principio, el cuerpo podría funcionar continuamente hasta el infinito. Observa que hacer una cosa generalmente disminuye la cantidad de cosas por hacer.

ejercicio

```
Un mago junior ha elegido un número secreto. Lo ha escondido en una variable
llamada secret_number. Quiere que todos los que ejecutan su programa jueguen el
juego Adivina el número secreto, y adivina qué número ha elegido para ellos.
¡Quiénes no adivinen el número quedarán atrapados en un bucle sin fin para siempre!
Desafortunadamente, él no sabe cómo completar el código.

Tu tarea es ayudar al mago a completar el código en el editor de tal manera que el
código:

Pedirá al usuario que ingrese un número entero.
Utilizará un bucle while.
Comprobará si el número ingresado por el usuario es el mismo que el número escogido
por el mago. Si el número elegido por el usuario es diferente al número secreto del
mago, el usuario debería ver el mensaje "¡Ja, ja! ¡Estás atrapado en mi bucle!"  y
se le solicitará que ingrese un número nuevamente. Si el número ingresado por el
usuario coincide con el número escogido por el mago, el número debe imprimirse en
la pantalla, y el mago debe decir las siguientes palabras: "¡Bien hecho, muggle!
Eres libre ahora".
¡El mago está contando contigo! No lo decepciones.


INFO EXTRA

Por cierto, observa la función print(). La forma en que lo hemos utilizado aquí se
llama impresión multilínea. Puede utilizar comillas triples para imprimir cadenas
en varias líneas para facilitar la lectura del texto o crear un diseño especial
basado en texto. Experimenta con ello.
```

```
secret_number = 777
```

```
print(
"""
+==================================+
| ¡Bienvenido a mi juego, muggle!  |
| Introduce un número entero       |
| y adivina qué número he          |
| elegido para ti.                 |
| Entonces,                        |
| ¿Cuál es el número secreto?      |
+==================================+
""")
num = 0

while num!=secret_number:
    num = int(input())
    if num!=secret_number: print("¡Ja, ja! ¡Estás atrapado en mi bucle!")
    else: print("¡Bien hecho, muggle! Eres libre ahora")
```

for

```
for i in range(100):
    # do_something()
    pass
```

```
for i in range(2, 8):
    print("El valor de i es actualmente", i)
```

- La palabra reservada for abre el bucle for; nota - No hay condición después de eso; no tienes que pensar en las condiciones, ya que se verifican internamente, sin ninguna intervención.
- Cualquier variable después de la palabra reservada for es la variable de control del bucle; cuenta los giros del bucle y lo hace automáticamente.
- La palabra reservada in introduce un elemento de sintaxis que describe el rango de valores posibles que se asignan a la variable de control.
- La función **range()** (esta es una función muy especial) es responsable de generar todos los valores deseados de la variable de control; en nuestro ejemplo, la función creará (incluso podemos decir que alimentará el bucle con) valores subsiguientes del siguiente conjunto: 0, 1, 2 .. 97, 98, 99; nota: en este caso, la función range() comienza su trabajo desde 0 y lo finaliza un paso (un número entero) antes del valor de su argumento.
- Nota la palabra clave **pass** dentro del cuerpo del bucle - no hace nada en absoluto; es una instrucción vacía : la colocamos aquí porque la sintaxis del bucle for exige al menos una instrucción dentro del cuerpo (por cierto, if, elif, else y while expresan lo mismo).

```
for i in range(2, 8, 3):
    print("El valor de i es actualmente", i)
```

- La función range() también puede aceptar tres argumentos: Echa un vistazo al código del editor.
- El tercer argumento es un incremento: es un valor agregado para controlar la variable en cada giro del bucle (como puedes sospechar, el valor predeterminado del incremento es 1).
- el conjunto generado por range() debe ordenarse en un orden ascendente. No hay forma de forzar el range() para crear un conjunto en una forma diferente. Esto significa que el segundo argumento de range() debe ser mayor que el primero.

ejercicio

```
¿Sabes lo que es Mississippi? Bueno, es el nombre de uno de los estados y ríos en los Estados Unidos. El río Mississippi tiene aproximadamente 2,340 millas de largo, lo que lo convierte en el segundo río más largo de los Estados Unidos (el más largo es el río Missouri). ¡Es tan largo que una sola gota de agua necesita 90 días para recorrer toda su longitud!

La palabra Mississippi también se usa para un propósito ligeramente diferente: para contar mississippily (mississippimente).

Si no estás familiarizado con la frase, estamos aquí para explicarte lo que significa: se utiliza para contar segundos.

La idea detrás de esto es que agregar la palabra Mississippi a un número al contar los segundos en voz alta hace que suene más cercano al reloj, y por lo tanto "un Mississippi, dos Mississippi, tres Mississippi" tomará aproximadamente unos tres segundos reales de tiempo. A menudo lo usan los niños que juegan al escondite para asegurarse de que el buscador haga un conteo honesto.


Tu tarea es muy simple aquí: escribe un programa que use un bucle for para "contar de forma mississippi" hasta cinco. Habiendo contado hasta cinco, el programa debería imprimir en la pantalla el mensaje final "¡Listos o no, ahí voy!"

Utiliza el esqueleto que hemos proporcionado en el editor.

INFO EXTRA

Ten en cuenta que el código en el editor contiene dos elementos que pueden no ser del todo claros en este momento: la sentencia import time y el método sleep(). Vamos a hablar de ellos pronto.

Por el momento, nos gustaría que supieras que hemos importado el módulo time y hemos utilizado el método sleep() para suspender la ejecución de cada función posterior de print() dentro del bucle for durante un segundo, de modo que el mensaje enviado a la consola se parezca a un conteo real. No te preocupes, pronto aprenderás más sobre módulos y métodos.
```

```python
import time

for count in range(1,6):
    print(count,"Mississippi")
    time.sleep(1)
print("¡Listos o no, ahí voy!")
```

break y continue

Hasta ahora, hemos tratado el cuerpo del bucle como una secuencia indivisible e inseparable de instrucciones que se realizan completamente en cada giro del bucle. Sin embargo, como desarrollador, podrías enfrentar las siguientes opciones:

Parece que no es necesario continuar el bucle en su totalidad; se debe abstener de seguir ejecutando el cuerpo

del bucle e ir más allá. Parece que necesitas comenzar el siguiente giro del bucle sin completar la ejecución del turno actual. Python proporciona dos instrucciones especiales para la implementación de estas dos tareas. Digamos por razones de precisión que su existencia en el lenguaje no es necesaria: un programador experimentado puede codificar cualquier algoritmo sin estas instrucciones. Tales adiciones, que no mejoran el poder expresivo del lenguaje, sino que solo simplifican el trabajo del desarrollador, a veces se denominan dulces sintácticos o azúcar sintáctica.

Estas dos instrucciones son:

- **break**: sale del bucle inmediatamente, e incondicionalmente termina la operación del bucle; el programa comienza a ejecutar la instrucción más cercana después del cuerpo del bucle.
- **continue**: se comporta como si el programa hubiera llegado repentinamente al final del cuerpo; el siguiente turno se inicia y la expresión de condición se prueba de inmediato.

Ambas palabras son palabras clave reservadas.

ejemplos

```
# break - ejemplo

print("La instrucción break:")
for i in range(1, 6):
    if i == 3:
        break
    print("Dentro del bucle.", i)
print("Fuera del bucle.")


# continue - ejemplo

print("\nLa instrucción continue:")
for i in range(1, 6):
    if i == 3:
        continue
    print("Dentro del bucle.", i)
print("Fuera del bucle.")
```

```
largest_number = -99999999
counter = 0

while True:
    number = int(input("Ingresa un número o escribe -1 para finalizar el programa:
"))
    if number == -1:
        break
    counter += 1
    if number > largest_number:
        largest_number = number

if counter != 0:
    print("El número más grande es", largest_number)
else:
    print("No has ingresado ningún número.")
```

```
largest_number = -99999999
counter = 0

number = int(input("Ingresa un número o escribe -1 para finalizar el programa: "))

while number != -1:
    if number == -1:
        continue
    counter += 1

    if number > largest_number:
        largest_number = number
    number = int(input("Ingresa un número o escribe -1 para finalizar el programa:
"))

if counter:
    print("El número más grande es", largest_number)
else:
    print("No has ingresado ningún número.")
```

ejercicio

```
La instrucción break se implementa para salir/terminar un bucle.

Diseña un programa que use un bucle while y le pida continuamente al usuario que
ingrese una palabra a menos que ingrese "chupacabra" como la palabra de salida
secreta, en cuyo caso el mensaje "¡Has dejado el bucle con éxito". Debe imprimirse
en la pantalla y el bucle debe terminar.

No imprimas ninguna de las palabras ingresadas por el usuario. Utiliza el concepto
de ejecución condicional y la sentencia break.
```

```
while True:
    text = input()
    if text == "chupacabra": break
print("¡Has dejado el bucle con éxito")
```

ejercicio

```
La sentencia continue se usa para omitir el bloque actual y avanzar a la siguiente
iteración, sin ejecutar las sentencias dentro del bucle.

Se puede usar tanto con while y for.

Tu tarea aquí es muy especial: ¡Debes diseñar un devorador de vocales! Escribe un
programa que use:

Un bucle for.
El concepto de ejecución condicional (if-elif-else).
La sentencia continue.
Tu programa debe:

Pedir al usuario que ingrese una palabra.
```

```
Utiliza user_word = user_word.upper() para convertir la palabra ingresada por el
usuario a mayúsculas; hablaremos sobre los llamados métodos de cadena y el upper()
muy pronto, no te preocupes.
Utiliza la ejecución condicional y la instrucción continue para "comer" las
siguientes vocales A , E , I , O , U de la palabra ingresada.
Imprime las letras no consumidas en la pantalla, cada una de ellas en una línea
separada.
```

```
user_word = input()
user_word = user_word.upper()

for letter in user_word:
    if letter=="A" or letter=="E" or letter=="I" or letter=="O" or letter=="U":
continue
    print(letter)
```

ejercicio

```
Tu tarea aquí es aún más especial que antes: ¡Debes rediseñar el devorador de
vocales (feo) del laboratorio anterior (3.1.2.10) y crear un mejor devorador de
vocales (bonito) mejorado! Escribe un programa que use:

Un bucle for.
El concepto de ejecución condicional (if-elif-else).
La instrucción continue.
Tu programa debe:

Pedir al usuario que ingrese una palabra.
Utilizar user_word = user_word.upper() para convertir la palabra ingresada por el
usuario a mayúsculas; hablaremos sobre los llamados métodos de cadena y el upper()
muy pronto, no te preocupes.
Emplea la ejecución condicional y la instrucción continue para "comer" las
siguientes vocales A , E , I , O , U de la palabra ingresada.
Asigne las letras no consumidas a la variable word_without_vowels e imprime la
variable en la pantalla.
Analiza el código en el editor. Hemos creado word_without_vowels y le hemos
asignado una cadena vacía. Utiliza la operación de concatenación para pedirle a
Python que combine las letras seleccionadas en una cadena más larga durante los
siguientes giros de bucle, y asignarlo a la variable word_without_vowels.
```

```
word_without_vowels = ""

user_word = input()
user_word = user_word.upper()

for letter in user_word:
    if letter=="A": continue
    elif letter=="E": continue
    elif letter=="I": continue
    elif letter=="O": continue
    elif letter=="U": continue
    else: word_without_vowels += letter
```

```
print(word_without_vowels)
```

El bucle while y la rama else

```
i = 1
while i < 5:
    print(i)
    i += 1
else:
    print("else:", i)
```

El bucle for y la rama else

```
for i in range(5):
    print(i)
else:
    print("else:", i)
```

ejercicio

```
Escucha esta historia: Un niño y su padre, un programador de computadoras, juegan
con bloques de madera. Están construyendo una pirámide.

Su pirámide es un poco rara, ya que en realidad es una pared en forma de pirámide,
es plana. La pirámide se apila de acuerdo con un principio simple: cada capa
inferior contiene un bloque más que la capa superior.

Tu tarea es escribir un programa que lea la cantidad de bloques que tienen los
constructores, y generar la altura de la pirámide que se puede construir utilizando
estos bloques.

Nota: La altura se mide por el número de capas completas: si los constructores no
tienen la cantidad suficiente de bloques y no pueden completar la siguiente capa,
terminan su trabajo inmediatamente.

Prueba tu código con los datos que hemos proporcionado.
```

```
blocks = int(input("Ingresa el número de bloques: "))

used_blocks = 0
height = 0

while blocks>=used_blocks:
    height += 1
    used_blocks = used_blocks + height
    if (used_blocks+height) >= blocks: break

print("La altura de la pirámide:", height)
```

ejercicio

```
En 1937, un matemático alemán llamado Lothar Collatz formuló una hipótesis
intrigante (aún no se ha comprobado) que se puede describir de la siguiente manera:

Toma cualquier número entero que no sea negativo y que no sea cero y asígnale el
nombre c0.
Si es par, evalúa un nuevo c0 como c0 Ã· 2.
De lo contrario, si es impar, evalúe un nuevo  c0  como 3 Ã c0 + 1.
Si c0 â  1, salta al punto 2.
La hipótesis dice que, independientemente del valor inicial de c0, el valor siempre
tiende a 1.

Por supuesto, es una tarea extremadamente compleja usar una computadora para probar
la hipótesis de cualquier número natural (incluso puede requerir inteligencia
artificial), pero puede usar Python para verificar algunos números individuales.
Tal vez incluso encuentres el que refutaría la hipótesis.

Escribe un programa que lea un número natural y ejecute los pasos anteriores
siempre que c0 sea diferente de 1. También queremos que cuente los pasos necesarios
para lograr el objetivo. Tu código también debe mostrar todos los valores
intermedios de c0.

Sugerencia: la parte más importante del problema es como transformar la idea de
Collatz en un bucle while- esta es la clave del éxito.
```

Operaciones lógicas y de bits en python: and, or not

- AND
- OR
- NOT
- Expresiones lógicas: Puedes estar familiarizado con las leyes de De Morgan. Dicen que:
  - La negación de una conjunción es la separación de las negaciones. → `not (p and q) == (not p) or (not q)`
  - La negación de una disyunción es la conjunción de las negaciones. → `not (p or q) == (not p) and (not q)`
- Operadores bit a bit
  - **&** (ampersand) - conjunción a nivel de bits. → requieres exactamente dos 1s para proporcionar 1 como resultado.
  - **|** (barra vertical) - disyunción a nivel de bits. → requiere al menos un 1 para proporcionar 1 como resultado.
  - **~** (tilde) - negación a nivel de bits.
  - **^** (signo de intercalación) - o exclusivo a nivel de bits (xor). → requiere exactamente un 1 para proporcionar 1 como resultado.
  - Agreguemos un comentario importante: los argumentos de estos operadores deben ser enteros. No debemos usar flotantes aquí.
- Operaciones lógicas frente a operaciones de bit:

```
i = 15 # 00000000000000000000000000001111
j = 22 # 00000000000000000000000000010110
print(i and j) # True
```

```
print(i & j) # 00000000000000000000000000000110 -> 6
```

- ¿Cómo tratamos los bits individuales?
  - La variable almacena la información sobre varios aspectos de la operación del sistema. Cada bit de la variable almacena un valor de si/no. También se te ha dicho que solo uno de estos bits es tuyo, el tercero (recuerda que los bits se numeran desde cero y el número de bits cero es el más bajo, mientras que el más alto es el número 31). Los bits restantes no pueden cambiar, porque están destinados a almacenar otros datos. Aquí está tu bit marcado con la letra x:

    ```
    flag_register = 0000000000000000000000000000x000
    ```

  - Es posible que tengas que hacer frente a las siguientes tareas:
    - Comprobar el estado de tu bit: deseas averiguar el valor de su bit; comparar la variable completa con cero no hará nada, porque los bits restantes pueden tener valores completamente impredecibles, pero puedes usar la siguiente propiedad de conjunción:

      ```
      x & 1 = x
      x & 0 = 0
      ```

    - Dicha secuencia de ceros y unos, cuya tarea es tomar el valor o cambiar los bits seleccionados, se denomina máscara de bits. Construyamos una máscara de bits para detectar el estado de tus bits. Debería apuntar a el tercer bit. Ese bit tiene el peso de 2^3=8. Se podría crear una máscara adecuada mediante la siguiente sentencia:

      ```
      the_mask = 8
      if flag_register & the_mask:
          # Mi bit se estableció en 1.
      else:
          # Mi bit se restableció a 0.
      ```

    - Reinicia tu bit: asigna un cero al bit, mientras que todos los otros bits deben permanecer sin cambios; usemos la misma propiedad de la conjunción que antes, pero usemos una máscara ligeramente diferente, exactamente como se muestra a continuación:

      ```
      11111111111111111111111111110111
      ```

    - Tenga en cuenta que la máscara se creó como resultado de la negación de todos los bits de la variable the_mask. Restablecer el bit es simple, y se ve así (elige el que más te guste):

      ```
      flag_register = flag_register & ~the_mask
      flag_register &= ~the_mask
      ```

    - Establece tu bit : asigna un 1 a tu bit, mientras que todos los bits restantes deben permanecer sin cambios; usa la siguiente propiedad de disyunción:

      ```
      x | 1 = 1
      x | 0 = x
      ```

      ```
      flag_register = flag_register | the_mask
      flag_register |= the_mask
      ```

    - Niega tu bit: reemplaza un 1 con un 0 y un 0 con un 1. Puedes utilizar una propiedad interesante del operador ~x:

      ```
      x ^ 1 = ~x
      x ^ 0 = x
      ```

```
flag_register = flag_register ^ the_mask
flag_register ^= the_mask
```

- Desplazamiento izquierdo binario y desplazamiento derecho binario:
  - Python ofrece otra operación relacionada con los bits individuales: shifting. Esto se aplica solo a los valores de número entero, y no debe usar flotantes como argumentos para ello.
  - Ya aplicas esta operación muy a menudo y muy inconscientemente. ¿Cómo multiplicas cualquier número por diez? Echa un vistazo:

    ```
    12345 × 10 = 123450
    ```

  - Como puede ver, multiplicar por diez es de hecho un desplazamiento de todos los dígitos a la izquierda y llenar el vacío resultante con cero.¿División entre diez? Echa un vistazo:

    ```
    12340 ÷ 10 = 1234
    ```

  - Dividir entre diez no es más que desplazar los dígitos a la derecha.
  - La computadora realiza el mismo tipo de operación, pero con una diferencia: como dos es la base para los números binarios (no 10), desplazar un valor un bit a la izquierda corresponde a multiplicarlo por dos; respectivamente, desplazar un bit a la derecha es como dividir entre dos (observe que se pierde el bit más a la derecha).

Los operadores de cambio en Python son un par de dígrafos: < < y > >, sugiriendo claramente en qué dirección actuará el cambio.

```
value << bits
value >> bits
```

listas