Clase 4 – Ciclos

IIC1103 Sección 9 – 2019-2

Profesor: Felipe López

03-09-2019

Resumen de la clase

```
if operación lógica:
        comando 1
else:
        comando 2
```

```
if operación lógica:
        if operación lógica:
                comando 1
        else:
                comando 2
else:
        comando 3
```

```
if operación lógica 1:
        comando 1
elif operación lógica 2:
        comando 2
…
elif operación lógica N:
        comando N
else:
        comando 2
```

Contenidos

1. `while`
2. `for`
3. Ejercicios

Recordemos este ejemplo de la clase pasada:

```
numero = int(input("Ingrese un número que sea mayor que 5\n"))

if numero>5:
    print("El número ingresado es mayor que 5.")
else:
    print("¡ERROR! El número que ingresaste no es mayor que 5.")
```

¿Qué podríamos hacer para verificar **siempre** que el usuario esté ingresando un número que es mayor que 5?

```
numero = int(input("Ingrese un número que sea mayor que 5\n"))

if numero>5:
    print("El número ingresado es mayor que 5.")
else:
    print("¡ERROR! El número que ingresaste no es mayor que 5.")
```

Una opción podría ser lo siguiente:

```
numero = int(input("Ingrese un número que sea mayor que 5\n"))

if numero>5:
    print("El número ingresado es mayor que 5.")
else:
    print("¡ERROR! El número que ingresaste no es mayor que 5.")

numero = int(input("Ingrese un número que sea mayor que 5\n"))

if numero>5:
    print("El número ingresado es mayor que 5.")
else:
    print("¡ERROR! El número que ingresaste no es mayor que 5.")
```

```
numero = int(input("Ingrese un número que sea mayor que 5\n"))

if numero>5:
    print("El número ingresado es mayor que 5.")
else:
    print("¡ERROR! El número que ingresaste no es mayor que 5.")

numero = int(input("Ingrese un número que sea mayor que 5\n"))

if numero>5:
    print("El número ingresado es mayor que 5.")
else:
    print("¡ERROR! El número que ingresaste no es mayor que 5.")
```

¿Y si el usuario se equivoca más de dos veces?

Ciclos

- Python posee herramientas para poder ejecutar el mismo código un determinado número de veces.
- Esto se hace a través del comando `while`.

Ciclos

- `while`, al igual que `if`, funciona con una operación lógica.
  - Se evalúa al "entrar" al ciclo, es decir, si la operación lógica se cumple, se ejecuta el ciclo por primera vez.
  - Si la operación lógica se cumple, o es verdadera, el código se repite.
  - Si la operación lógica es falsa, entonces el ciclo termina.
- La estructura general de un `while` es la siguiente:

```
while operación lógica:
    comando 1
```

Ciclos

Veamos un ejemplo:

```
while True:
    print("Hola")
```

Ciclos

Veamos un ejemplo:

```
while True:
    print("Hola")
```

Hay que responder dos preguntas:

- ¿Se va a ejecutar el ciclo?

Sí, porque la operación lógica es verdadera

- ¿Cuántas veces se ejecutará?

Infinitas veces, porque la operación lógica siempre será verdadera.

Ciclos

Veamos un ejemplo:

```
while False:
    print("Hola")
```

Hay que responder dos preguntas:

- ¿Se va a ejecutar el ciclo?
  No, porque la operación lógica es falsa.
- ¿Cuántas veces se ejecutará?
  0 veces, ya que la operación lógica es falsa y nunca pasará a ser verdadera.

Ciclos

- Veamos otro ejemplo. En este caso, queremos imprimir en consola los números del 1 al 20.
- Escribir 20 veces `print("1"), print("2")`, y así sucesivamente claramente no es eficiente.
- Por lo tanto, ocuparemos un ciclo para poder hacerlo de una forma más fácil.

Ciclos

```
variable_numerica = 1

while variable_numerica <= 20:
    print(str(variable_numerica))
    variable_numerica = variable_numerica + 1

print("terminó el conteo")
```

Ciclos

```
variable_numerica = 1

while variable_numerica <= 20:
    print(str(variable_numerica))
    variable_numerica = variable_numerica + 1

print("terminó el conteo")
```

Podemos volver a responder estas dos importantes preguntas.

¿Se va a ejecutar el ciclo?
Sí, porque la operación lógica es verdadera. Su valor es 1 y como 1 es menor o igual que 20 entonces sí se cumple

¿Cuántas veces se ejecutará?
20 veces, dado que la operación lógica se cumple mientras `variable_numerica` sea menor o igual que 20.
Cuando el valor de esta es igual a 21, entonces la operación lógica ya no se cumple y el ciclo se deja de ocupar.

Ciclos

- Para que quede completamente claro, iremos paso a paso:

```
variable_numerica = 1

while variable_numerica <= 20:
    print(str(variable_numerica))
    variable_numerica = variable_numerica + 1

print("terminó el conteo")
```

1. Se asigna un 1 a `variable_numerica`
2. Se evalúa la operación lógica; 1 es menor o igual que 20, lo que es verdadero, por lo tanto se entra "dentro" del ciclo
3. Se imprime el 1 en la consola
4. `variable_numerica` aumenta en 1, ahora es 2.
5. Se evalúa la operación lógica; 2 es menor o igual que 20, lo que es verdadero, por lo tanto se entra "dentro" del ciclo.
6. Se imprime el 2 en la consola
7. `variable_numerica` aumenta en 1, ahora es 3.
8. (… se repiten los pasos 5, 6 y 7 hasta que `variable_numerica` es 19)
9. Se evalúa la operación lógica; 19 es menor o igual que 20, lo que es verdadero, por lo tanto se entra "dentro" del ciclo.
10. Se imprime el 19 en la consola
11. `variable_numerica` aumenta en 1, ahora es 20.
12. Se evalúa la operación lógica; 20 es menor o igual que 20, lo que es verdadero, por lo tanto se entra "dentro" del ciclo.
13. Se imprime el 20 en la consola
14. `variable_numerica` aumenta en 1, ahora es 21.
15. Se evalúa la operación lógica; 21 NO es menor o igual que 20. La operación lógica es falsa por lo tanto no entra dentro del ciclo.
16. Se imprime en consola "terminó el conteo".

Ciclos

- Hay veces que no sabemos a priori cuándo debería terminar el ciclo.
- En general se ocupa cuando esperamos que el usuario ingrese cierta información o cuando la ejecución misma del código cambie el resultado de la operación lógica.
- Por ejemplo, escribamos un programa que imprima en consola un texto hasta que el usuario indique lo contrario.

Ciclos

Por ejemplo, escribamos un programa que imprima en consola un texto hasta que el usuario indique lo contrario.

```
input_del_usuario = input("Escribe 1 para que termine el programa")

while input_del_usuario != "1":
    input_del_usuario = input("Escribe 1 para que termine el programa")
```

Ciclos

Si por alguna razón extra a la operación lógica de un `while` uno quisiera terminarlo, uno puede ocupar el comando break.

Por ejemplo,

```
while True:
    print("test")
    break

print("test2")
```

En este ejemplo, la operación lógica es verdadera (y siempre lo es ya que no varía). Al entrar a la primera iteración del ciclo se imprime en consola “test”. Sin embargo, y por el comando `break` se termina el `while` y se imprime en consola “test2”.

break

5 min

```
numero = int(input("Ingrese un número que sea mayor que 5\n"))

if numero>5:
    print("El número ingresado es mayor que 5.")
else:
    print("¡ERROR! El número que ingresaste no es mayor que 5.")

numero = int(input("Ingrese un número que sea mayor que 5\n"))

if numero>5:
    print("El número ingresado es mayor que 5.")
else:
    print("¡ERROR! El número que ingresaste no es mayor que 5.")
```

¿Y si el usuario se equivoca más de dos veces?

```
numero=int(input("Ingresa un número mayor que 5"))

while numero<=5:
 numero=int(input("¡Error! El número que ingresaste no es mayor
que 5. Por favor ingrésalo nuevamente"))

print("El número",numero,"ingresado es mayor que 5 :)")
```

for

- Existe un caso especial del `while`, denominado `for`.
- Una de las ventajas de este comando es que tiene integrado un índice que va aumentando, desde un límite inferior a un límite superior.
- Sirve especialmente para poder ejecutar un código un cierto número de veces.
- La estructura general de un `for` es:

```
for i in range(a,b):
    comando 1
```

Donde:

- `i` es la variable que incrementa
- `a` es el valor inicial de `i`
- `i` aumenta hasta `b-1`

for

- Tomando uno de los ejemplos anteriores:

```
variable_numerica = 1

while variable_numerica <= 20:
    print(str(variable_numerica))
    variable_numerica = variable_numerica + 1

print("terminó el conteo")
```

```
for variable_numerica in range(1,21):
    print(str(variable_numerica))
```

Se puede observar que se cumple lo mismo que el ejemplo del `while` anterior. En este caso, `variable_numerica` parte en el valor 1 y aumenta hasta llegar a 20.

Ejercicio

- Hackerrank

https://www.hackerrank.com/s2-contest-3

Este no es un *contest* o desafío del curso, sino que se ocupó como ejemplo. No se preocupen que el año corresponda al 2016.

Los ejercicios que vamos a ver son “Monitoreo de traslado” y “La suerte de los reyes”.

Ejercicio 1: Monitoreo de Traslado

- Una empresa de taxis quiere incorporar en sus automóviles un dispositivo que notifique a sus pasajeros la distancia restante a su destino. Dado un recorrido de largo *l* y una distancia de muestreo *m*, este dispositivo le avisará al pasajero la distancia que falta para llegar al destino cada *m* metros. Así al recorrer los primeros metros, se desplegará un mensaje del estilo ***te faltan l-m metros para llegar a tu destino***, al recorrer los siguientes *m* metros, se desplegará un mensaje del estilo ***te faltan l-2m metros para llegar a tu destino***, y así sucesivamente.
- Escribe un programa en Python que reciba la distancia *l* (en metros) del recorrido, y el nivel de muestreo *m* (en metros). Tu programa debe imprimir cada *m* metros recorridos la distancia restante para llegar al destino. Cuando el pasajero llega a su destino, el programa debe imprimir ***has llegado a tu destino***
- Puede asumir que los valores ingresados son enteros positivos.

Ejercicio 1: Monitoreo de Traslado

```
l=int(input())
m=int(input())
l-=m

while l>=0:
    print("te faltan",l,"metros para llegar a tu destino")
    l-=m
print("has llegado a tu destino")
```

Ejercicio 2: La suerte de los reyes

- Cuatro amigos aburridos y con mucho tiempo libre inventaron un juego llamado "La suerte de los reyes". Este se juega con una baraja de naipes, pero utilizando solamente números del 1 al 9 (por simplicidad, el "As" se representa como un 1), cuatro reyes y una reina.
- El objetivo del juego es tener el menor puntaje posible. Por turnos cada jugador saca un naipe. El número de la carta será el puntaje que obtiene en la ronda. Si un jugador saca un *rey*, el puntaje que lleva acumulado hasta ese momento se multiplica por dos. Si un jugador saca la *reina*, su puntaje disminuye en 12 por cada rey que haya salido en el juego hasta ese momento. Después de que se saca la reina, se acaba el juego y se hace el recuento de puntaje. El jugador con menos puntaje es el ganador.
- Escribe un programa en Python que simule un juego de "La suerte de los reyes" entre cuatro jugadores. El input del programa representará la baraja de cartas, y cada linea indicará el naipe que saca cada uno de los jugadores en su turno, comenzando por el *jugador 1*, luego el *jugador 2*, *jugador 3* y *jugador 4*. De acuerdo a las reglas del juego, tu programa debe imprimir el jugador ganador de la partida. Puedes asumir que siempre habrá **solamente un ganador**, es decir, no existe el empate.

Ejercicio 2: La suerte de los reyes

- El input se ve de la siguiente forma:

  naipe 1 (int o string)

  naipe 2 (int o string)

  ...

  naipe n (int o string)

- Los valores de input pueden ser:
  - enteros del 1 al 9
  - string *rey*
  - string *reina*
  - Solamente habrá un ganador

Ejercicio en clases

- Escribe un programa donde un usuario ingrese un número *n.*
  - El número n **debe ser** mayor que 0.
  - El programa debe imprimir todos los números entre 1 y *n.*
  - Cuando se imprima un número par se debe imprimir esto en consola y cuando sea impar también.

P3 I1 2018-2

Pregunta 3

"Ruleta DCC" es el juego más exitoso de la tradicional "Fonda de Don Yadran", que organiza el Departamento de Ciencia de la Computación (DCC). Por supuesto, funciona en Python. Desafortunadamente se perdió el código, así es que te piden que lo vuelvas a crear.

Para participar, hay que apostar 100, por lo que el juego parte con 100 puntos. La ruleta genera solamente números aleatorios entre 1 y 36. El juego termina si sale 7 (en ese caso, quien estaba jugando se lleva el puntaje acumulado), si sale 13 (en ese caso, pierde todo), o si se le terminan los puntos.

Cuando empiezan a salir los números:

- Si detectas una secuencia de 3 números pares, se suma al puntaje todos los números de ruleta (no importa si es 7 o 13, ni si hay secuencias de números pares o impares) que salgan hasta que aparece uno de los tres números pares (el que también se suma). En el siguiente turno, la secuencia de números se reinicia.
- Si detectas una secuencia de 3 números impares, se resta al puntaje la suma de los siguientes 5 números de ruleta que salgan (no importa si es 7 o 13, ni si hay secuencias de números pares o impares). En el siguiente turno, la secuencia de números se reinicia.

Lo que debes hacer es implementar un programa que simule el juego "Ruleta DCC". A continuación se muestran algunos ejemplos del funcionamiento del programa:

Resumen de la clase

```
while operación lógica:
    comando 1
```

- Se evalúa al “entrar” al ciclo, es decir, si la operación lógica se cumple, se ejecuta el ciclo por primera vez.
- Si la operación lógica se cumple, o es verdadera, el código se repite.
- Si la operación lógica es falsa, entonces el ciclo termina.

```
for i in range(a,b):
    comando 1
```

Donde:

- `i` es la variable que incrementa
- `a` es el valor inicial de `i`
- `i` aumenta hasta `b-1`

Links

- https://repl.it/@FelipeLopez/IIC1103Ciclos que contiene todos los ejemplos de la clase.

Bibliografía

- http://runest.ing.puc.cl/while.html
- A. B. Downey. Think Python: How to think like a computer scientist. Green Tea Press, 2013 -> Capítulo 7

Clase 4 – Ciclos

IIC1103 Sección 9 – 2019-2

Profesor: Felipe López

03-09-2019