Control de Flujo: ciclos

Clase #06

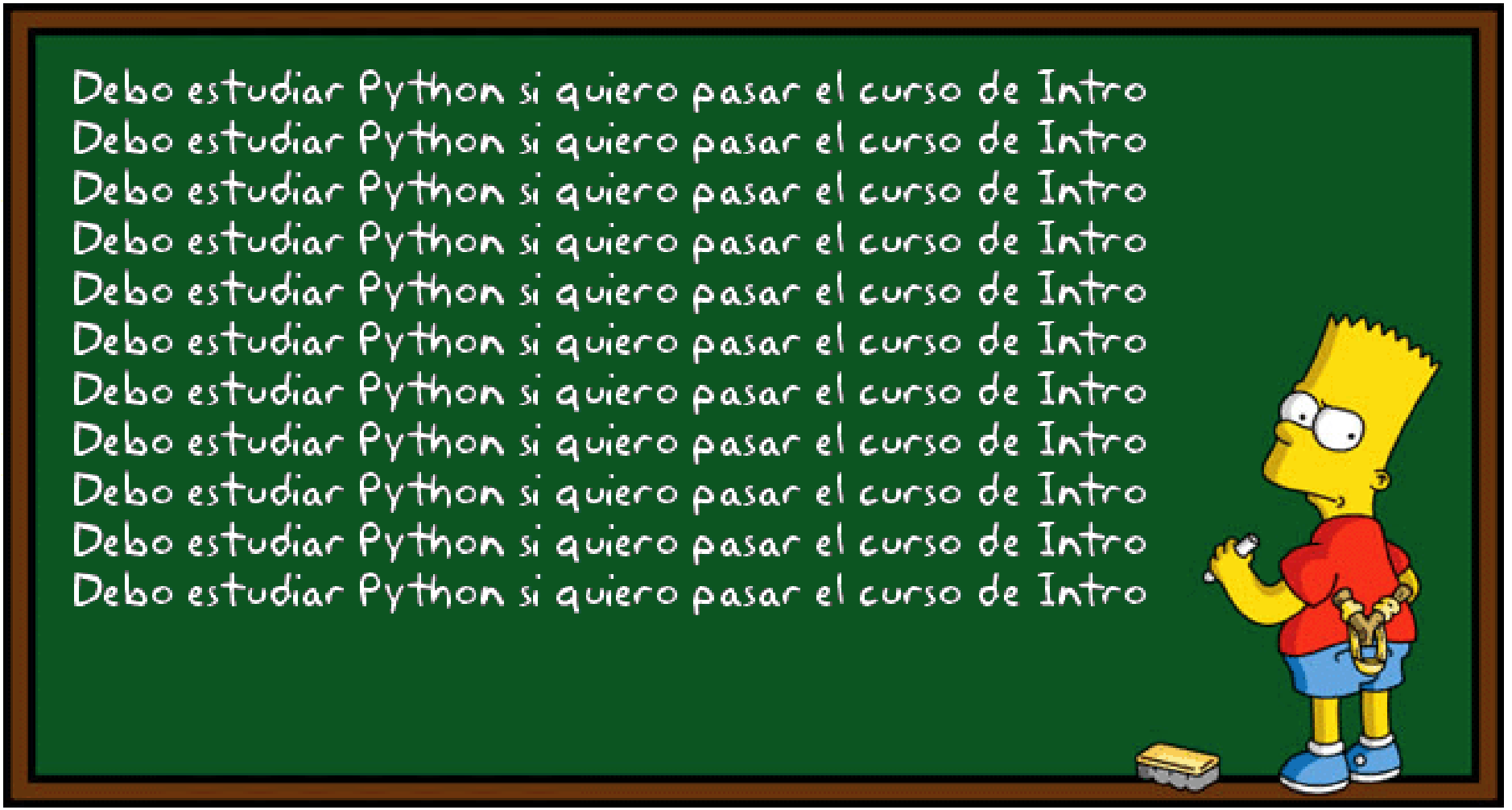
IIC1103 – Introducción a la Programación

Marcos Sepúlveda (marcos@ing.puc.cl)

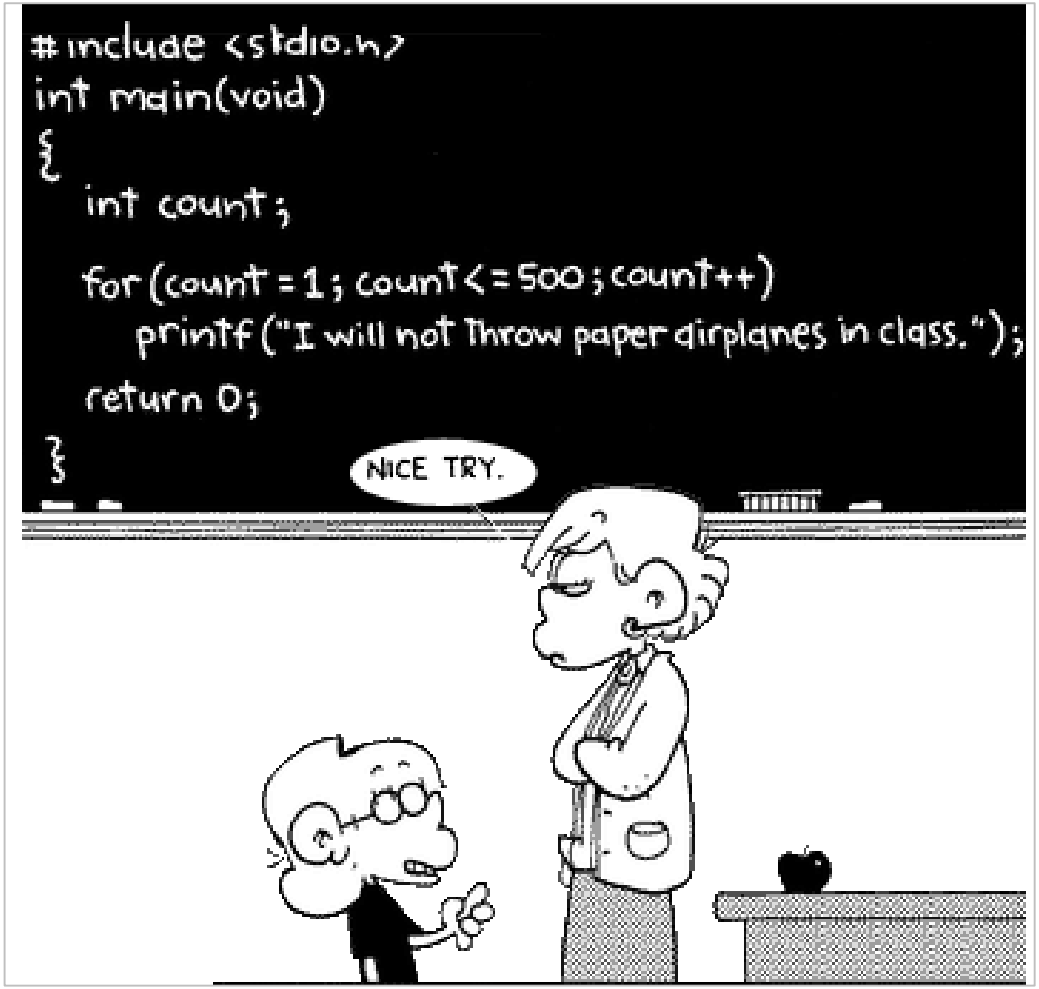
Contenido

- Control de flujo:
  - Instrucción **`for`**
  - Contadores y acumuladores
  - Ciclos anidados
- Comparación de **`while`** y **`for`**
- Ejercicios

Ciclos – ¿para qué servían?



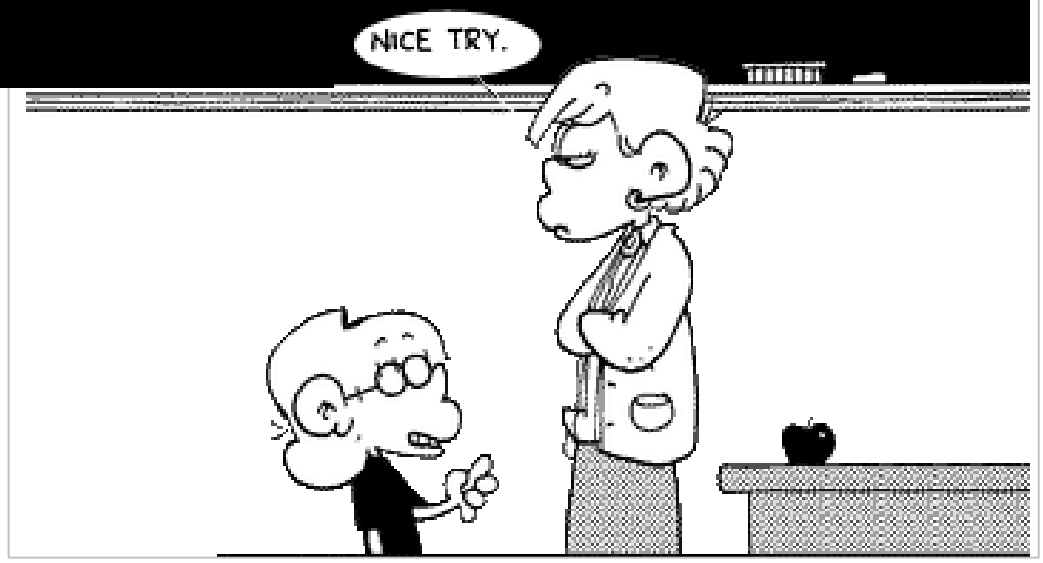
Ciclos – ¿para qué servían?



Nota: esta imagen sólo es un ejemplo de iteración en otro lenguaje de programación (en C)

Ciclos – en Python con `while`

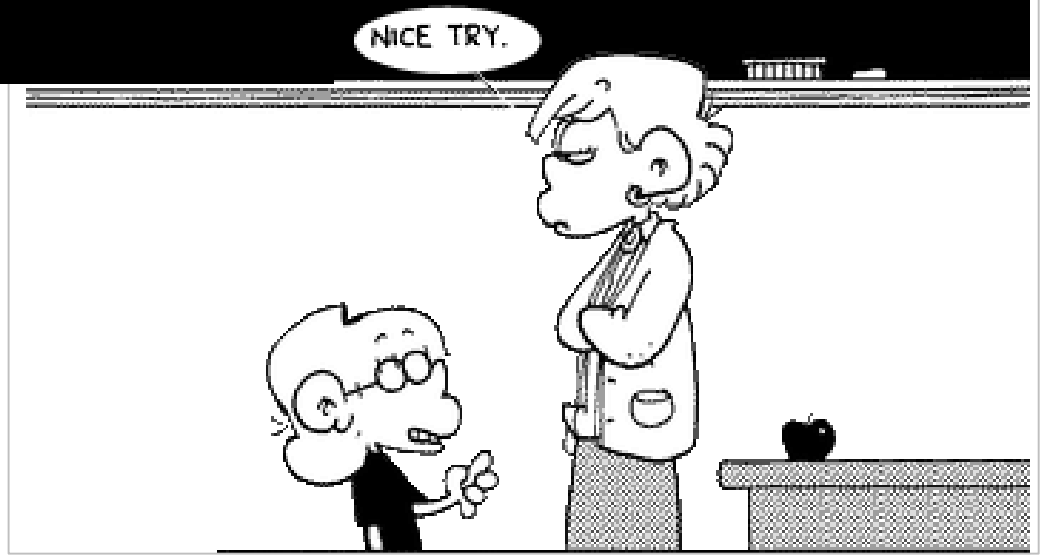
```
i = 0
while i < 100:
    print("Debo estudiar Python si quiero pasar el curso de Intro")
    i += 1
```



Nota: parodiando el anterior, pero en Python

Ciclos – en Python con `for`

```
for i in range(0,100):
    print("Debo estudiar Python si quiero pasar el curso de Intro")
```



Nota: parodiando el anterior, pero en Python

Otra forma de crear ciclos – `for`

- Hemos visto que es posible repetir un conjunto de instrucciones utilizando **`while`**

```
i = 0
while i < 100:
    print("Debo estudiar Python si quiero pasar el curso de Intro")
    i += 1
```

- Ahora veremos otra sentencia para realizar ciclos, la instrucción de iteración **`for`**

```
for i in range(0,100):
    print("Debo estudiar Python si quiero pasar el curso de Intro")
```

Iteración: `for`

- Sintaxis

```
for variable in range():
    instrucción
    ...
```

- El conjunto de instrucciones se ejecuta tantas veces como se establezca en el tipo de dato **`range`**.
- Nota: las instrucciones que estén dentro del **`for`** deben estar indentadas.

¿Qué significa `range(n)`?

- Una forma de saberlo es probándolo en Python:

```
print("Valores de for con range(5)")
for i in range(5):
    print("En esta iteración, i vale", i)

print("Valores de for con range(2,7)")
for i in range(2,7):
    print("En esta iteración, i vale", i)

print("Valores de for con range(0,5)")
for i in range(0,5):
    print("En esta iteración, i vale", i)
```

¿Qué significa `range(n)`? (con 1 argumento)

- **`range`** es un tipo de dato que crea una lista inmutable de números enteros en sucesión aritmética

- **`range(n)`** crea una lista creciente de **`n`** términos enteros que empieza en **`0`** y acaba antes de llegar a **`n`** (los términos aumentan de uno en uno)
  - **`range(n) == [0, 1, ... , n-1]`**

- Ejemplo:
  - **`range(5)`** toma los valores **`[0,1,2,3,4]`**

¿Qué significa `range(n,m)`? (con 2 argumentos)

- **`range`** es un tipo de dato que crea una lista inmutable de números enteros en sucesión aritmética

- **`range(n,m)`** crea una lista creciente de **`m-n`** términos enteros que empieza en **`n`** y acaba antes de llegar a **`m`** (los términos aumentan de uno en uno)
  - **`range(n,m) == [n, n+1, ... , m-1]`**

- Ejemplo:
  - **`range(2,7)`** toma los valores **`[2,3,4,5,6]`**

Variables en ciclos (`while` o `for`)

Dos conceptos que nos servirán en las iteraciones son:

- **Contadores**
  - Se entiende por **contador** una variable que lleva la cuenta del número de veces que se ha cumplido una condición

- **Acumuladores**
  - Se entiende por **acumulador** una variable que acumula el resultado de una operación

Contador – ejemplo

- Escribir un programa que cuente cuántos múltiplos de 7 hay entre 1 y 1000

```
cuenta = 0

for i in range(1,1001):
    if i%7 == 0:
        cuenta += 1

print("Desde 1 a 1000, hay", cuenta, "múltiplos de 7")
```

```
>>>
Desde 1 a 1000, hay 142 múltiplos de 7
```

Acumulador – ejemplo

- Escribir un programa que pida 3 sueldos y entregue la suma total de los mismos

```
sueldo_total = 0

for i in range(3):
    sueldo = int(input("Ingrese sueldo: "))
    sueldo_total += sueldo

print("Sueldo total es: ", sueldo_total)
```

```
>>>
Ingrese sueldo: 222
Ingrese sueldo: 333
Ingrese sueldo: 444
Sueldo total es:  999
```

Contador y Acumulador – ejemplo

- Escribir un programa que cuente y sume los múltiplos de 13 que están entre 500 y 2587

```
cuenta = 0
suma = 0

for i in range(500,2588):
    if i%13 == 0:
        cuenta += 1
        suma += i

print("Cantidad de múltiplos de 13 es:", cuenta)
print("Suma de múltiplos de 13 es:", suma)
```

```
>>>
Cantidad de múltiplos de 13 es: 161
Suma de múltiplos de 13 es: 249067
```

Iteración – ciclos anidados

```
i = 0
while i < n:
    j = 0
    while j < m:
        instrucción
        ...
        j += 1
    i += 1
```

```
for i in range(n):
    for j in range(m):
        instrucción
        ...
```

- Un ciclo anidado es un ciclo situado en el cuerpo de otro ciclo.
- Por lo tanto, las instrucciones que están más adentro se ejecutarán `n * m` veces

Ciclos anidados – ejemplo

- Escribir un programa que muestre la tabla de multiplicar de 1 a 10

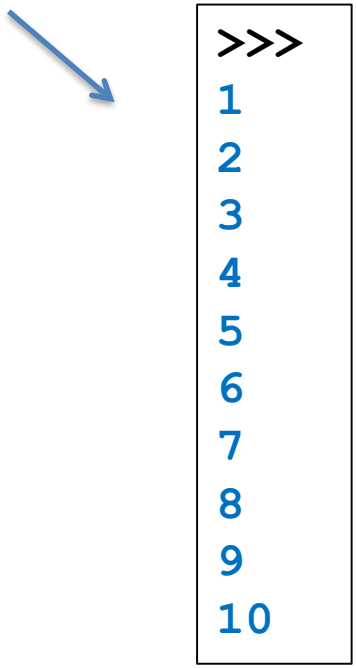
```
for i in range(1,11):
    print("La tabla de:", i)
    for j in range(1, 11):
        print(i, "*", j, "=", i*j)
```

```
>>>
La tabla de: 1
1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
1 * 4 = 4
1 * 5 = 5
1 * 6 = 6
1 * 7 = 7
1 * 8 = 8
1 * 9 = 9
1 * 10 = 10
La tabla de: 2
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
2 * 6 = 12
2 * 7 = 14
2 * 8 = 16
2 * 9 = 18
2 * 10 = 20
...
```

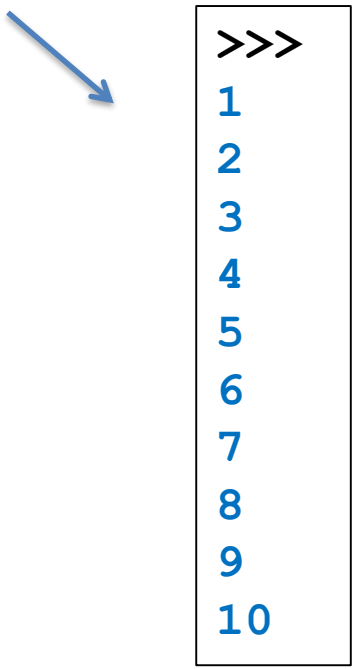
¿Cuál de los dos usar? `while` o `for`

¿Cuál de los dos usar? `while` o `for`

```
i = 1
while i <= 10:
    print(i)
    i += 1
```




```
for i in range(1,11):
    print(i)
```



¿Cuál de los dos usar? `while` o `for`

```
i = 1
while i <= 10:
    print(i)
    i += 1
```

```
for i in range(1,11):
    print(i)
```

- Crear ciclos indefinidos hasta que cierta condición se cumpla en el programa

¿Cuál de los dos usar? `while` o `for`

```
i = 1
while i <= 10:
    print(i)
    i += 1
```

```
for i in range(1,11):
    print(i)
```

- Comúnmente para ejecutar líneas de código un número determinado de veces.

Ejercicio – número perfecto

- Un número natural se dice perfecto si es igual a la suma de sus divisores propios (divisores menores que el número).

- Por ejemplo, los números 6 y 28 son perfectos, ya que:
  - Los divisores propios de 6 son 1, 2 y 3, y 6 == (1 + 2 + 3)
  - Los divisores propios de 28 son 1, 2, 4, 7 y 14, y 28 == (1 + 2 + 4 + 7 + 14)

- A su vez, el número 12 no es perfecto, ya que:
  - Los divisores propios de 12 son 1, 2, 3, 4 y 6, y 12 != (1 + 2 + 3 + 4 +6)

- Haga un programa que pida un número al usuario y le indique si el número recibido es o no perfecto.

Ejercicio – número perfecto

```
n = int(input("Ingrese número: "))

suma = 0

i = 1
while i <= n//2:
    if n % i == 0:
        suma += i
    i += 1

if n == suma:
    print(n, "es un número perfecto")
else:
    print(n, "no es un número perfecto")
```

```
>>>
Ingrese número: 28
28 es un número perfecto
```

Algunos tips de edición

Obligatorio

- 4 espacios por nivel de indentación (en el IDLE se puede redefinir la cantidad de espacios de un TAB)
- No mezclar tabs + espacios

Sugerencias

- Añadir espacio después de “,” cuando se escribe texto en `print()`
- Añadir espacio después de “:”; nunca antes.