Control de flujo II: Estructuras de iteración

IEEE
UPCT SB

Iteración

- La iteración es la repetición de un bloque de código para lograr un resultado.
- El número de repeticiones puede depender de un valor o una expresión.
- La iteración nos permite:
  - Realizar la misma operación con distintos valores. Ej: determinar primos <20.
  - Realizar una misma operación a la salida de la anterior operación. Ej: método de newton.
  - Aplicar un método general a una lista de casos particulares. Ej: extraer info string.

Bucle while

- Funciona como un if que repite el bloque mientras se cumpla la expresión.
- Su sintaxis:

```
while expresión:
    # Repetición
```

- Ej:

```
a = 10
while a > 0:
    print(a)
    a -= 1
```

```
ctrl = True
while ctrl:
    edad = input()
    ctrl = edad.isdecimal()
    if ctrl:
        print("NaN, repite.")
edad = int(edad)
```

Bucle for

- En el bucle for iteramos sobre un conjunto de valores que se van asignando a una variable.
- La sintaxis es:
```
for var in iterable:
    # Repetición
```
- Ej:
```
for i in [1,2,3,4]:
    print(i)
```
- Ej:
```
for c in “Hola Mundo!”:
    print(c)
```

Funciones para for

- Existen un grupo de funciones muy útiles para el bucle for. Algunas de ellas son:
  - La función range. Genera una lista de números ordenados.

    Admite dos sintaxis:
    - range(stop)
    - range(init, stop[, step]

    Ej:
```
for i in range(5):
    print(i)
```

break y continue

- Existen dos palabras reservadas que pueden cambiar el funcionamiento de una iteración. Estas son break y continue.
- break para una iteración y sale del bucle.
- continue para una iteración y continua con la siguiente.
- Ej:

```
while(True):
    a = int(input())
    if a > 0:
        break
```

```
for i in range(100):
    if i % 2 != 0:
        continue
    # Cosas pares
```

Bucles anidados

- Se pueden anidar tantos bucles como queramos pero hay que tener en cuenta que podemos afectar a la eficiencia del programa.
- Ej:

```
max_vocal = 0
for linea in texto.split('\n'):
  for palabra in linea.split(' '):
    contador_de_vocales = 0
    for c in palabra:
      if c in 'aeiouáéíóú':
        contador_de_vocales += 1
    if contador_de_vocales > max_vocal:
      max_vocal = contador_de_vocales
      palabra_max_vocales = palabra
```

Ejercicio

- Utiliza el algoritmo babilónico para calcular la raíz cuadrada de un número introducido por el usuario con una precisión de 4 dígitos.

  El algoritmo babilónico es un algoritmo por iteración que permite calcular raíces cuadras intentando asemejar el área de un rectángulo a un cuadrado. Su expresión matemática es:

$$f_o(x) = x \qquad f_n(x) = \frac{1}{2} \cdot \left(\frac{x}{f_{n-1}(x)} + f_{n-1}(x) \right) \qquad f_\infty(x) = \sqrt{x}$$

Ejercicio

- Realiza un programa que tras introducir un polinomio real de grado n encuentre la raíz más cercana a 0.
- Recomendación: El método de Newton es un potente método iterativo.
- AMPLIACIÓN PARA CASA (no obligatorio pero suma): mejora la búsqueda y encuentra todas las raíces usando el teorema de Bolzano.

Ejercicio

- Vamos a mejorar el *parser* de la sesión anterior. Imaginemos que recibimos un texto como este de una estación meteorológica cada hora:

  “hora:14-temp:297.5-hum:78.4-rain:0”

  Siendo temp → K, hum → %Rel, rain → mm/h

- Determina:
  - Temperatura y humedad máxima y mínima y a qué hora se dieron.
  - Precipitación acumulada durante el día.
- Obtén los datos de las siguiente diapositiva.

Ejercicio

```
hora:0-temp:297.5-hum:78.4-rain:0
hora:1-temp:296.3-hum:79.2-rain:0
hora:2-temp:294.1-hum:78.1-rain:0
hora:3-temp:292.5-hum:78.0-rain:0
hora:4-temp:291.8-hum:78.6-rain:0
hora:5-temp:292.4-hum:80.4-rain:10
hora:6-temp:294.7-hum:82.4-rain:11
hora:7-temp:295.5-hum:88.4-rain:40
hora:8-temp:298.6-hum:92.4-rain:42
hora:9-temp:300.5-hum:96.4-rain:44
hora:10-temp:301.2-hum:95.6-rain:20
hora:11-temp:302.2-hum:95.7-rain:22
```

Ejercicio

```
hora:12-temp:302.9-hum:95.4-rain:10
hora:13-temp:302.9-hum:95.4-rain:12
hora:14-temp:303.2-hum:92.2-rain:6
hora:15-temp:303.4-hum:90.1-rain:2
hora:16-temp:302.5-hum:88.0-rain:0
hora:17-temp:300.8-hum:88.6-rain:0
hora:18-temp:299.4-hum:88.4-rain:0
hora:19-temp:297.7-hum:89.4-rain:0
hora:20-temp:294.5-hum:87.4-rain:0
hora:21-temp:291.6-hum:89.4-rain:0
hora:22-temp:290.5-hum:90.4-rain:16
hora:23-temp:290.2-hum:93.6-rain:21
```

for con unpacking

- Se puede iterar con más de un valor mediante un proceso llamado desempaquetamiento. Este proceso disuelve conjuntos de datos:

```
for i,j in [(1,'a'),(2,'b'),(3,'c')]:
    print(i,j)
```

- Algunas funciones devuelven valores que se pueden desempaquetar.

```
for indice, valor in enumerate("Hey you!"):
    print(indice, valor)
```

Unpacking básico

- El desempaquetamiento es algo más complejo y tiene que ver con las estructuras de datos. Lo veremos en detalle más adelante.
- Sí podemos adelantar que también podemos desempaquetar en las operaciones de asignación:

```
a, b, c = 0, 0, 0

name, subname = input(“Name: ”),
input(“Surname: ”)
```