4. ESTRUCTURAS DE CONTROL

Programación I

Grado en Inteligencia Artificial

Curso 2022/2023

Contenidos

- Programación estructurada
- Estructura condicional o selectiva
  - Sentencia **if**
  - Estructura **if / elif / else**
- Estructura iterativa o repetitiva
  - Bucle **while**
  - Bucle **for – in**
- Gestión de excepciones

Programación estructurada

- Un programa sencillo se puede crear en base a un patrón *secuencial* puro: Entrada de datos → procesamiento → presentación de resultados
- Teorema de la **programación estructurada**: “es posible construir cualquier programa mediante la combinación de las estructuras de control *secuencial*, *selectiva* y *repetitiva*”

Expresiones lógicas

- Las estructuras selectiva y repetitiva se basan en la evaluación de condiciones que combinan **operadores relacionales y lógicos**:

Operador	Comparación
==	Igual a
!=	Distinto de
<	Menor que
<=	Menor o igual que
>	Mayor que
>=	Mayor o igual que

Operador	Operación
not	Negación
and	Conjunción
or	Disyunción

Estructura selectiva

- **Ejemplo**: ecuaciones de primer grado

```
#Resolución de la ecuación a·x + b = 0
a = float(input('Valor de a: '))
b = float(input('Valor de b: '))

x = -b / a

print('Solución: ', x)
```

- ¿Y si el coeficiente *a* vale 0?

Estructura selectiva

- En Python se usa la palabra reservada **if**

```
if condición:
    acción
    acción
    ...
    acción
```

- Las sentencias que forman el *cuerpo* del condicional se deben *sangrar* respecto a la línea que contiene la condición

Ejemplo mejorado

```
#Resolución de la ecuación a·x + b = 0
a = float(input('Valor de a: '))
b = float(input('Valor de b: '))

if a != 0:
    x = -b / a
    print('Solución: ', x)
```

Ejemplo (aún más) mejorado

```
#Resolución de la ecuación a·x + b = 0
a = float(input('Valor de a: '))
b = float(input('Valor de b: '))

if a != 0:
    x = -b / a
    print('Solución: ', x)
if a == 0:
    print('La ecuación no tiene solución')
```

Anidamiento de condicionales

```
#Resolución de la ecuación a·x + b = 0
a = float(input('Valor de a: '))
b = float(input('Valor de b: '))

if a != 0:
    x = -b / a
    print('Solución: ', x)
if a == 0:
    if b != 0:
        print('La ecuación no tiene solución')
    if b == 0:
        print('Resultado indeterminado')
```

Condicionales con alternativas

```
if condición:
    acciones
else:
    otras acciones
```

```
if condición:
    acciones
elif otra condición:
    otras acciones
else:
    últimas acciones
```

Código definitivo

```
#Resolución de la ecuación a·x + b = 0
a = float(input('Valor de a: '))
b = float(input('Valor de b: '))

if a != 0:
    x = -b / a
    print('Solución: ', x)
else:
    if b != 0:
        print('La ecuación no tiene solución')
    else:
        print('Resultado indeterminado')
```

Evaluación con cortocircuitos

```
if a == 0 or 1/a > 1:
    acciones
```

La segunda condición sólo se evalúa si la primera devuelve False

```
if a != 0 and 1/a > 1:
    acciones
```

La segunda condición sólo se evalúa si la primera devuelve True

En ninguno de los dos casos se hará una división por 0

Estructura repetitiva

- Popularmente conocida como *bucle*
- Atienden a la necesidad de repetir la ejecución de ciertas instrucciones en los programas
- Dos alternativas en Python:
  - Sentencia **while**
  - Sentencia **for – in**

Bucle while

```
while condición:
    acción
    acción
    ...
    acción
```

Las instrucciones que forman el *cuerpo* del bucle deben sangrarse adecuadamente

¿Qué valor toma la variable **i** justo después del bucle?

```
i = 0
while i < 10:
    print(i)
    i += 1
print('Hecho')
```

Cálculo de un sumatorio

```
sumatorio = 0
i = 1
while i <= 1000:
    sumatorio += i
    i += 1
print(sumatorio)
```

¿Producen el mismo resultado?

```
sumatorio = 0
i = 0
while i < 1000:
    i += 1
    sumatorio += i
print(sumatorio)
```

Importancia de inicializar las variables

Requisitos en la entrada

```
from math import sqrt

x = -1
while x < 0:
    x = float(input('Un n° positivo? '))

c = chr(8730) #Unicode 0x221A
out = '{0}({1})={2}'.format(c, x, sqrt(x))
print(out)
```

Bucle for – in

```
for valor in serie:
    acción
    acción
    ...
```

Python incluye diversas estructuras de datos para gestión de colecciones por las que se puede *iterar*: cadenas, listas, rangos…

```
lista = ['Pepe', 'Ana', 'Juan']
for nom in lista:
    print('Hola, {0}'.format(nom))
```

```
for i in range(1, 6):
    print(i)
```

```
for c in 'Pepe':
    print(c)
```

Función range()

- Genera un objeto *enumerador*
- Se suele invocar con 2 parámetros enteros que establecen los límites
- Si sólo se pasa 1 parámetro se considera que el primer valor del rango es 0
- Si se llama con 3 parámetros el tercero establece el incremento (1 por defecto)

Comparación entre bucles

```
suma = 0
i = 1
while i <= 100:
    suma += i
    i += 1
print(suma)
```

```
suma = 0
for i in range(1, 101):
    suma += i
print(suma)
```

En los bucles que siguen este patrón, la variable de control **i** suele recibir el nombre de índice

¿Qué ventajas e inconvenientes presenta cada versión?

Elección del bucle apropiado

- El bucle **while** es el adecuado cuando no se conoce de antemano el número de iteraciones que se van a realizar
- El bucle **for** es más apropiado cuando el número de repeticiones necesarias está determinado a priori
- Sección 4.2.7 del libro de Marzal et al.: *elaboración* de un programa que determina si un número dado es primo

Versión con for

```
num = int(input('Su número? '))
if num > 1:
    es_primo = True
    for divisor in range(2, num):
        if num % divisor == 0:
            es_primo = False
else:
    es_primo = False

if es_primo == True: #simplificar!
    print('Su número es primo')
else: print('Su número no es primo')
```

Versión con while

```
num = int(input('Su número? '))
if num > 1:
    es_primo = True
    divisor = 2
    while divisor < num:
        if num % divisor == 0:
            es_primo = False
        divisor += 1
else: es_primo = False

if es_primo: print('Su número es primo')
else: print('Su número no es primo')
```

Versión con while mejorada

```
num = int(input('Su número? '))
if num > 1:
    es_primo = True
    divisor = 2
    while divisor < num and es_primo:
        if num % divisor == 0:
            es_primo = False
        divisor += 1
else: es_primo = False

if es_primo: print('Su número es primo')
else: print('Su número no es primo')
```

Ruptura de bucles con break

```
num = int(input('Su número? '))
if num > 1:
    es_primo = True
    divisor = 2
    while divisor < num:
        if num % divisor == 0:
            es_primo = False
            break
        divisor += 1
else: es_primo = False
if es_primo: print('Su número es primo')
else: print('Su número no es primo')
```

Ruptura de bucles con break

```
num = int(input('Su número? '))

if num > 1:
    es_primo = True
    for divisor in range(2, num):
        if num % divisor == 0:
            es_primo = False
            break
else: es_primo = False

if es_primo: print('Su número es primo')
else: print('Su número no es primo')
```

Anidamiento de bucles

```
lim = int(input('Su número? '))

for num in range(2, lim+1):
    es_primo = True
    for divisor in range(2, num):
        if num % divisor == 0:
            es_primo = False
            break
    if es_primo: print(num, end=' ')
print()
```

¿Qué bucle rompe este **break**?

Dentro de un **for** no se debe cambiar el índice

Gestión de excepciones

- Tipos de errores:
  - Sintácticos
  - En tiempo de ejecución: **excepciones**
  - Lógicos
- La estructura de control **if** aporta un medio para la gestión de **excepciones**
- Python aporta una estructura de control específica que aligera el código, evitando la proliferación de comprobaciones

Estructura try – except

```
try:
    acción potencialmente errónea
    acción potencialmente errónea
    ...
    acción potencialmente errónea
except:
    acción para tratar el error
    acción para tratar el error
    ...
    acción para tratar el error
```

Ejemplo de uso de try – except

```
a = float(input('Valor de a: '))
b = float(input('Valor de b: '))

try:
    x = -b/a
    print('Solución: ', x)
except:
    if b != 0:
        print('No tiene solución')
    else:
        print('Tiene infinitas soluciones')
```

Discriminación de excepciones

```
from math import sqrt
a = float(input('Valor de a: '))
b = float(input('Valor de b: '))
c = float(input('Valor de c: '))
try:
    x1 = (-b + sqrt(b**2 - 4*a*c)) / (2 * a)
    x2 = (-b - sqrt(b**2 - 4*a*c)) / (2 * a)
    if x1 == x2: print('Solución (doble):', x1)
    else: print('Soluciones:', x1, 'y', x2)
except ZeroDivisionError:
    if b != 0: print('No tiene solución')
    else: print('Tiene infinitas soluciones')
except ValueError: print('No hay soluciones
reales')
```