ESTRUCTURAS DE CONTROL DO FLUXO

Ada Lovelace (1815-1852)

- Inglaterra, século XIX (1815-1852)
- Inventora do primeiro programa de ordenador
- Precursora da informática actual hai máis de 100 anos
- Linguaxe de programación de sistemas de tempo real chamado ADA na súa honra.

Estructuras de control do fluxo

- Tipos de estructuras de control:
  - Sentenzas de selección.
  - Sentenzas de iteración.
- **Indentación**: é semellante a sangría na linguaxe escrita de textos.
  - Moitas linguaxes de programación utilizan a indentación para incrementar a lexibilidade do código.
  - En **Python**, **a indentación é obrigatoria** para implementar as estruturas de fluxo.
  - Unha indentación de 4 (catro) espazos en branco, indicará que as instruccións indentadas forman parte dunha mesma estructura de control.

Estruturas de control do fluxo

- Unha estructura de control defínese como:

```
inicio da estructura de control:
    expresións
```

- **Python 2.x : Códificación**: en python 2.x a codificación do texto realizase por defecto en ASCII (7 bits). Para codificar o texto en Unicode necesítase engadir unha directiva ó comezo dun arquivo Python, co fin de indicar ó sistema, a codificación de caracteres utilizada no arquivo (utf-8, podería ser calquera codificación de caracteres, que se non se pon o programa podería producir erros con caracteres estraños como a ñ).
  - `# -*- coding: utf-8 -*-`
- **Python 3.x realiza por defecto a codificación en Unicode polo que non se necesita incluir ningunha directiva para utilizar caracteres como a ñ no código.**

Sentenzas de selección

- As estructuras de control condicionais ou **sentenzas de selección** permiten avaliar se unha ou máis condición se cumpren para decidir que bloque de código se vai executar. A avaliación de condicións realízase con operadores relacionais ou lóxicos e devolven dous posibles resultados: verdadeiro ou falso (`True` ou `False`).
- Na definición de sentenzas de selección utilizanse tres palabras claves de python: `if`, `elif` e `else`.
- Tipos de sentencias de selección:
  - Bloque `if`
  - Bloque `if:else`
  - Bloque `if:elif:else`

Operadores relacionais

Operadores binarios que actúan sobre valores numéricos e devolven un valor lóxico

Operador	Descrición	Exemplo
==	¿Son iguais?	r= 5 == 3 # r vale False
!=	¿Son distintos?	r= 5 != 3 # r vale True
<	¿é menor?	r= 5 < 3 # r vale False
<=	¿é menor ou igual?	r= 5 <= 5 # r vale True
>	¿é maior?	r= 5 > 3 # r vale True
>=	¿é maior ou igual?	r= 5 >= 3 # r vale True

Operadores lóxicos

Operadores binarios ou unarios que operan sobre valores lóxicos e devolven un valor lóxico

Operador	Descrición	Exemplo
and	>>>a and b ¿cúmprese a e b?	r= True and True # r vale True # resto de casos r vale False
or	>>> a or b ¿cúmprese a ou b?	r= False and False # r vale False # resto de casos r vale True

not	>>> not a
	Non a
		r= not True # r vale False
		r= not False # r vale True

Precedencia de operadores

Cando unha expresión contén operadores aritméticos, relacionais e lóxicos execútanse primeiro as operacións aritméticas, seguidas polas relacionais e logo as lóxicas (Detalles na táboa extraida do libro de Andrés Marcial e Isabel Gracia). Pódese modificar esta precedencia utilizando parénteses.

Operación	Operador	Aridad	Asociatividad	Precedencia
Exponenciación	**	Binario	Por la derecha	1
Identidad	+	Unario	—	2
Cambio de signo	-	Unario	—	2
Multiplicación	*	Binario	Por la izquierda	3
División	/	Binario	Por la izquierda	3
Módulo (o resto)	%	Binario	Por la izquierda	3
Suma	+	Binario	Por la izquierda	4
Resta	-	Binario	Por la izquierda	4
Igual que	==	Binario	—	5
Distinto de	!=	Binario	—	5
Menor que	<	Binario	—	5
Menor o igual que	<=	Binario	—	5
Mayor que	>	Binario	—	5
Mayor o Igual que	>=	Binario	—	5
Negación	**not**	Unario	—	6
Conjunción	**and**	Binario	Por la izquierda	7
Disyunción	**or**	Binario	Por la izquierda	8

Tabla 2.4: Características de los operadores Python. El nivel de precedencia 1 es el de mayor prioridad.

Sentencia de selección

Bloque if

```
if expresion:
    Liña_1
    Liña_2

Liña_3
```

Valórase a expresión.

No caso de que sexa verdadeira (**True**) execútase o bloque de código dentro do **if** (Linea_1 e Linea_2), por último procesarase a Linea_3.

Se a expresion é falsa (False), omítese a execución do bloque **if** e o programa continuará na sentenza seguinte, no noso caso Linea_3.

Bloque if:else

```
if expresion:
    se_certo_isto
else:
    se_falso_isto
```

No caso do **if:else**, cando a expresión non é verdadeira execútanse as sentenzas baixo o **else** (neste caso **se_falso_isto**)

Sentenzas de selección

Bloque if:elif

```
if expresion1:
    sentenzas1
elif expresion2:
    sentenzas2
elif expresion3:
    sentenzas3
elif expresion4:
    sentenzas4
.
.
.
else:
    se_falso_isto
```

Valórase a expresión1. Se é verdadeira, executase sentenzas1 e remátase.

Se expresión1 é False, avalíase expresion2: se é True, executase sentenzas2 e remátase.

Se expresión2 tamén é False, avalíase expresion3: se é True, execútase sentenzas3 e remátase.

Se expresion3 tamén é False, avalíase expresion4: se é True, execútanse sentenzas4 e remátase.

E así sucesivamente: se ningunha das expresionN é True, execútase se_falso_isto. Este bloque else é optativo (pode non estar)

Se temos *N* condicións, teremos *N* posibles execucións (N+1 se hai bloque **else**).

Sentenzas de iteración

- As estruturas de control de iteración ou **sentenzas de iteración** (tamén chamadas bucles, lazos ou ciclos) permiten executar o mesmo código, de xeito repetido, mentres se cumpra unha condición ou un certo número de veces.
- Tipos de sentencias de iteración:
  - Bucle **`while`** (mentres se cumpra unha condición)
  - Bucle **`for`** (un certo número de veces)

Bucle while

- Repite todo o bloque de código do bucle mentres a expresión avaliada sexa verdadeira (True). Sintaxe:

```
while condición:
    Liña_1
    Liña_2

Liña_3
```

Avalíase a condición (que debe ser unha expresión lóxica)

Antes de comezar, hai que darlle valores ás variábeis que interveñen na condición

Se se cumpre, execútanse as sentenzas, e volve a avaliarse a condición.

```
i=1
while i != -1:
      i=input("i? ")
print "fin"
```

Repítese a execución das sentenzas ata que se deixa de cumprir a condición

As sentenzas deben modificar as variábeis que interveñen na condición para que esta deixe de cumprirse nalgún momento

Le por teclado números ata introducir -1

Bucle for

- A sentenza for permite iterar sobre os valores dunha secuencia.

```
for x in secuencia:
    sentenzas
```

- Se queremos que a variable tome valores entre 0 ata un número n, utilizar a función **`range(n)`** que devolve unha lista que comeza en 0 e remata en n-1
  - >>> **`range(4)`** # devolveria [0,1,2,3]
- range(inicio, fin, incremento) permite construir listas de números que comecen en inicio e rematen en fin sendo a diferencia entre elementos consecutivos de incremento.
  - >>> **`range(2,20,5)`** # devolveria [2, 7, 12, 17]

Exemplos de bucle for

- Iteración normal:

```
for i in range(n):
    print('%d'%i)
```

- Iteración sobre os elementos dunha lista ou array:

```
x=[5,7,3,2]
for i in x:
    print('%d'%i)
```

- Iteración sobre os índices e elementos dunha lista:

```
x=[5,7,3,2]
for i, y in enumerate(x):
    print('%d %d'%(i, y))
```

- Iteración sobre os elementos de dúas listas:

```
x=[5,7,3,2]; y=[5,3,6,8]
for i, j in zip(x, y):
    print('%d %d'%(i, j))
```

Impresión de listas/vectores

- O máis básico:

  x=[1,2,3] ou x=array([1,2,3])

  print('x=',x) -> Imprime x=[1,2,3]

- Impresión vectorizada:

  print(' '.join(str(i) for i in x)) -> Imprime: 1 2 3

  print(' '.join('%i'%i for i in x)) -> Igual

- Impresión dunha matriz:

  a=array([[1,2,3],[4,5,6]])

  print(a)

Operación sobre listas con **for**

- Para realizar unha operación sobre unha lista x=[‘1’,’2’,’3’]
- Exemplo: transformar a lista **x** nunha lista **y** de enteiros:
- y=[int(i) for i in x]
- Isto obtén a lista y=[1,2,3]
- Pódese xeralizar a outras operacións sobre tódolos elementos dunha lista
- Alternativa: usar vectores de **numpy** e vectorizar

Break and Continue

- Tanto os bucle **while** como **for** ofrecen a posibilidade de modificar o fluxo do programa coas sentenzas **`break`** e **`continue`**.
- **`break`**: sae do bucle while ou for.
- **`continue`**: salta ó inicio do bucle ignorando o resto do código do bucle e volvendo a avaliar a expresión do bucle.