PYTHON

Módulo 4 – Control de flujo: bucles y decisiones

MÓDULO 4: CONTROL DE FLUJO: BUCLES Y DECISIONES

Introducción

MÓDULO 4: CONTROL DE FLUJO: BUCLES Y DECISIONES

- El control de flujo permite establecer caminos alternativos de ejecución o repetir la ejecución bloques de código en función de condiciones lógicas.
- Se conocen como **sentencias compuestas.**
  - Sentencia if
  - Sentencia while
  - Sentencia for
  - Sentencia try
  - Sentencia with

https://docs.python.org/es/3/reference/compound_stmts.html

MÓDULO 4: CONTROL DE FLUJO: BUCLES Y DECISIONES

- El operador ==
  - Compara dos operandos
  - Devuelve True si son iguales, False si son distintos
  - Tiene baja prioridad (primero se resuelven los operandos)
- El operador !=
  - Compara dos operandos
  - Devuelve True si son distintos, False si son iguales
  - Tiene baja prioridad (primero se resuelven los operandos)

MÓDULO 4: CONTROL DE FLUJO: BUCLES Y DECISIONES

- Otros operadores de comparación:
  - > Mayor que
  - < Menor que
  - >= Mayor o igual que (mayor no estricto)
  - <= Menor o igual que (menor no estricto)
  - Tienen la misma prioridad que == y !=

MÓDULO 4: CONTROL DE FLUJO: BUCLES Y DECISIONES

- Operadores lógicos:
  - and (alta prioridad)
  - or (baja prioridad)
  - not (alta prioridad)

MÓDULO 4: CONTROL DE FLUJO: BUCLES Y DECISIONES

- Precedencia de operadores:

Operator	Description
`(expressions...)`, `[expressions...]`, `{key: value...}`, `{expressions...}`	Binding or parenthesized expression, list display, dictionary display, set display
`x[index]`, `x[index:index]`, `x(arguments...)`, `x.attribute`	Subscription, slicing, call, attribute reference
`await x`	Await expression
`**`	Exponentiation [5]
`+x`, `-x`, `~x`	Positive, negative, bitwise NOT
`*`, `@`, `/`, `//`, `%`	Multiplication, matrix multiplication, division, floor division, remainder [6]
`+`, `-`	Addition and subtraction
`<<`, `>>`	Shifts
`&`	Bitwise AND
`^`	Bitwise XOR
`|`	Bitwise OR
`in`, `not in`, `is`, `is not`, `<`, `<=`, `>`, `>=`, `!=`, `==`	Comparisons, including membership tests and identity tests
`not x`	Boolean NOT
`and`	Boolean AND
`or`	Boolean OR
`if – else`	Conditional expression
`lambda`	Lambda expression
`:=`	Assignment expression

- https://docs.python.org/3/reference/expressions.html#operator-precedence

MÓDULO 4: CONTROL DE FLUJO: BUCLES Y DECISIONES

if

MÓDULO 4: CONTROL DE FLUJO: BUCLES Y DECISIONES

- if (instrucción condicional)

Forma 1:

```
if condición:
        bloque de código
```

Forma 2:

```
if condición:
        bloque de código
else:
        bloque de código
```

https://docs.python.org/3/reference/compound_stmts.html#the-if-statement

MÓDULO 4: CONTROL DE FLUJO: BUCLES Y DECISIONES

- if (instrucción condicional)

Forma 3:

```
if condición:
        bloque de código
elif condición:
        bloque de código
```

Forma 4:

```
if condición:
        bloque de código
elif condición:
        bloque de código
else:
        bloque de código
```

MÓDULO 4: CONTROL DE FLUJO: BUCLES Y DECISIONES

- if (instrucción condicional)

If anidados:

```
if condicion1:
        bloque de código
        if condicion2:
                bloque de código
elif condicion3:
        bloque de código
        if condicion4:
                bloque de código
```

MÓDULO 4: CONTROL DE FLUJO: BUCLES Y DECISIONES

- if (instrucción condicional)
- Reglas:
  - No se puede utilizar else sin if.
  - El bloque else es opcional.
  - El bloque else siempre va al final.
  - Si hay un bloque else, al menos una de las ramas de la estructura de decisión es ejecutada.
  - Si no hay un bloque else, puede que no se ejecute ninguna de las ramas de la estructura de decisión.
  - No se puede utilizar elif sin if.
  - Los bloques de código condicionados van sangrados con tabulador o espacios (normalmente 4 y no se pueden mezclar).
  - Admite listas de sentencias: if a>5: print("hola"); print("adios")

MÓDULO 4: CONTROL DE FLUJO: BUCLES Y DECISIONES

Operador ternario

MÓDULO 4: CONTROL DE FLUJO: BUCLES Y DECISIONES

- valor_true if condición else valor_false
- Sintaxis de expresión condicional tradicional:

```
if valor>5:
    resultado = True
else:
    resultado = False
```

- Sintaxis de expresión condicional usando operador ternario:

```
resultado = True if valor>5 else False
```

MÓDULO 4: CONTROL DE FLUJO: BUCLES Y DECISIONES

while

MÓDULO 4: CONTROL DE FLUJO: BUCLES Y DECISIONES

- Bucle while
  - Repite el bloque de código mientras se cumpla la condición

```
while condición:
        bloque de código
```

  - Si no hay condición de salida se produce un bucle infinito

```
while True:
        bloque de código
```

  - Si la condición no se cumple en la primera evaluación, el bloque de código no se ejecuta ninguna vez.
  - El bloque de código puede contener a su vez otras estructuras condicionales o de repetición.
  - Admite listas de sentencias.

https://docs.python.org/3/reference/compound_stmts.html#the-while-statement

MÓDULO 4: CONTROL DE FLUJO: BUCLES Y DECISIONES

for

MÓDULO 4: CONTROL DE FLUJO: BUCLES Y DECISIONES

- Bucle for
  - Repite un número determinado de veces el código contenido en el bloque.
  - Sintaxis: for item in colección.
    - Se puede utilizar slicing
  - Sintaxis: for índice in range()
    - Rangos: range(stop). Comienza en 0.
    - Rangos: range(start, stop)
    - Rangos: range(start, stop, step)
      - Sólo acepta enteros.
      - Start: incluido.
      - Stop: excluido.

https://docs.python.org/3/reference/compound_stmts.html#the-for-statement

MÓDULO 4: CONTROL DE FLUJO: BUCLES Y DECISIONES

- Bucle for
  - Indicando el número de iteracciones:

```
for i in range(100):
    bloque de código
```

Indicando un rango:

```
for i in range(1,100):
    bloque de código
```

Indicando un rango y un paso:

```
for i in range(1,100,2):
    bloque de código
```

MÓDULO 4: CONTROL DE FLUJO: BUCLES Y DECISIONES

- Bucle for
  - Creación de lista utilizando bucles for:

```
lista = [x for x in range(1,10,2)]
     [1, 3, 5, 7, 9]

lista2 = [x for x in lista[2:4]]
     [5, 7]
```

MÓDULO 4: CONTROL DE FLUJO: BUCLES Y DECISIONES

match

MÓDULO 4: CONTROL DE FLUJO: BUCLES Y DECISIONES

- match (equivalente a switch en otros lenguajes)
  - Nuevo a partir de la versión 3.10 del lenguaje.

```
edad = 19
match edad:
    case 0:
        print("0")
    case 19:
        print("19")
    case 25:
        print("25")
    case other:#other o _
        print("No sé")
```

https://docs.python.org/3/reference/compound_stmts.html#the-match-statement

MÓDULO 4: CONTROL DE FLUJO: BUCLES Y DECISIONES

Otras características

MÓDULO 4: CONTROL DE FLUJO: BUCLES Y DECISIONES

- Instrucciones de control de bucles:
  - break
    - Detiene la ejecución del bucle
  - continue
    - Detiene la ejecución de la iteración actual del bucle continuando en la siguiente iteración
  - pass
    - Permite construir estructuras sin funcionalidad sintácticamente correctas:

```
>>> for a in [1,2]:
...     pass
```

MÓDULO 4: CONTROL DE FLUJO: BUCLES Y DECISIONES

- Bloques de sentencias:
  - Separadas por ;

```
if a>3:
    print(a);print(a*2)
```

- while+else
- for+else
- El bloque else no se ejecuta si se hace break.
- Uso combinado en el control de flujo

MÓDULO 4: CONTROL DE FLUJO: BUCLES Y DECISIONES

- Recorrer dos iteradores simultáneamente:
  - Función zip(mezcla dos iteradores)

```
>>> dias = ["lunes","martes","miercoles"]
>>> valores = (3,4,5)
>>> for d,v in zip(dias,valores):
...         print(d,v)
...
lunes 3
martes 4
miercoles 5
>>>
```

MÓDULO 4: CONTROL DE FLUJO: BUCLES Y DECISIONES

try

MÓDULO 4: CONTROL DE FLUJO: BUCLES Y DECISIONES

- Control de errores: try-except
  - Concepto de exception
  - Permiten controlar errores previstos:

```
try:
    bloque de código
except:
    bloque de código
```

  - Tipos de excepciones:
    - https://docs.python.org/3/library/exceptions.html

https://docs.python.org/3/reference/compound_stmts.html#the-try-statement

MÓDULO 4: CONTROL DE FLUJO: BUCLES Y DECISIONES

- Control de errores: try-except
  - Especificiando el error:

```
try:
     bloque de código
except ZeroDivisionError:
     bloque de código
```

  - Especificiando el error, múltiples niveles:

```
try:
     bloque de código
except ZeroDivisionError:
     bloque de código
except:
     bloque de código
```

MÓDULO 4: CONTROL DE FLUJO: BUCLES Y DECISIONES

- Control de errores: try-except
  - Especificiando el error, múltiples excepciones, múltiples niveles:

```
try:
     bloque de código
except (ZeroDivisionError, ValueError):
     bloque de código
except:
     bloque de código
```

MÓDULO 4: CONTROL DE FLUJO: BUCLES Y DECISIONES

- Control de errores: try-except
  - Bloque else: se ejecuta si no se produce ningún error.

```
try:
     bloque de código
except:
     bloque de código
else:
     bloque de código
```

MÓDULO 4: CONTROL DE FLUJO: BUCLES Y DECISIONES

- Control de errores: try-except
  - Bloque finally: se ejecuta siempre.

```
try:
     bloque de código
except:
     bloque de código
finally:
     bloque de código
```

MÓDULO 4: CONTROL DE FLUJO: BUCLES Y DECISIONES

- Control de errores: try-except
  - Combinación

```
try:
     bloque de código
except:
     bloque de código
else:
     bloque de código
finally:
     bloque de código
```

MÓDULO 4: CONTROL DE FLUJO: BUCLES Y DECISIONES

with

Módulo 4: Control de flujo: bucles y decisiones

- with
  - Facilita la realización de operaciones habituales:
    - Manejo de ficheros.
    - Conexiones de red.
    - Bloqueos.
    - Acceso a bases de datos.
    - …
    -

https://docs.python.org/3/reference/compound_stmts.html#the-with-statement

MÓDULO 4: CONTROL DE FLUJO: BUCLES Y DECISIONES

- with con ficheros
- Sin with:

```
try:
    f = open("files_with_with.py", "r")
    info = f.read()
    print(info)
except:
    error("Ha ocurrido un error")
finally:
    f.close()
```

- Con with:

```
with open("files_with_with.py", "r") as f:
    info = f.read()
    print(info)
```