Facultad de Ciencias de la Administración

TECNICATURA UNIVERSITARIA EN DESARROLLO WEB

PROGRAMACIÓN I

Unidad V – Estructuras de Control

Ejecución secuencial de sentencias y Est. de control condicionales

Tecnicatura Universitaria en Desarrollo Web

Facultad de Ciencias de la Administración

Universidad Nacional de Entre Ríos

- **Objetivos**
  - Identificar las distintas alternativas de las que se dispone para controlar el flujo de ejecución de programas.
  - Entender como hacer combinaciones de las mismas.
  - Comprender cómo pueden diseñarse/documentarse algoritmos a través de diagramas.
- **Temas a desarrollar:**
  - Ejecución Secuencial de sentencias.
  - Estructuras de control condicionales: if, elif, else, match.
  - Estructuras de control iterativas: while, for, for in range, break, continue. Análisis de eficiencia.
  - Diagramas de flujo.

Estructuras de control

- *Una **estructura de control** es una sentencia de control y las sentencias cuya ejecución ésta controla.*
- Es un bloque de código que permite **agrupar** instrucciones de manera controlada.
- Todos los lenguajes de programación tienen un conjunto mínimo de instrucciones que permiten especificar el control propio del algoritmo que se requiere implementar.
  - Este conjunto mínimo debe contener estructuras para la ***Secuencia, Decisión, Selección e Iteración***.
- Secuencia
  - La estructura de control más simple está representada por una sucesión de instrucciones (por ejemplo asignaciones), en la que el ***orden de ejecución coincide con el orden físico de aparición de las instrucciones***.
  - La implementación de la estructura de control secuencia se logra simplemente ubicando ***una instrucción debajo de la anterior***.

Decisión - Ejecución condicional

- Cuando programamos, casi siempre necesitamos controlar condiciones y cambiar el comportamiento del programa de acuerdo a si se cumplen o no. Las estructuras de control condicionales nos dan esta habilidad.

```
if x > 0:
    print('x es positivo')
```

- La **expresión booleana** después de la palabra reservada `if` es llamada ***condición***. Si es `True` (verdadero), la sentencia identada que le sigue se ejecuta. Caso contrario, nada sucede.
- El `if` tiene la misma estructura que la definición de una función: un encabezado seguido por un cuerpo identado. Las sentencias como esta se llaman ***declaraciones compuestas***.
  - No hay límite para el número de sentencias que pueden aparecer en el cuerpo de un bloque `if`, pero tiene que ser al menos una.
  - Ocasionalmente, es útil tener un cuerpo sin sentencias (para guardar espacio para algo que vamos a agregar más tarde).
  - En ese caso, podemos escribir la palabra reservada `pass` que no hace nada.

```
if x < 0:
    pass # Aquí falta manejar qué pasa cuando x es menor que 0.
```

Ejecución alternativa

- Una segunda forma de la sentencia **`if`** es la ***ejecución alternativa***, donde hay dos posibilidades y una condición que determina cuál se ejecuta.
- La sintaxis es así:

```
if x % 2 == 0:
    print ('x es par')
else:
    print('x es impar')
```

- El resto de la división entera de **`x`** por **`2`** es **`0`** cuando **`x`** es par. En ese caso el programa muestra el mensaje: **x es par**.
- En caso que la condición sea **`False`** (falso), el segundo conjunto de sentencias se ejecuta.
- Dado que la condición debe ser **`True`** o **`False`**, exactamente una de las alternativas se ejecuta. Las alternativas son llamadas ***ramas***, porque como si fueran ramas donde se bifurca el flujo de ejecución.

Condiciones encadenadas

- A veces hay más de dos posibilidades y necesitamos ***más de dos ramas***. Una forma de expresar un cómputo de este tipo es encadenar condiciones.

```
if x < y:
    print ('x es menor que y')
elif x > y:
    print ('x es mayor que y')
else:
    print('x e y son iguales')
```

- `elif` es una abreviación de "**else if**".
  - Otra vez, exactamente ***sólo una de las ramas se va a ejecutar***.
  - No hay límite para la cantidad de sentencias que pueden aparecer dentro de un bloque `elif`.
  - Si hay una cláusula `else`, tiene que estar al final, no es obligatorio que exista.

Condiciones encadenadas (2)

```
if opcion == 'a':
    dibujar_a()
elif opcion == 'b':
    dibujar_b()
elif opcion == 'c':
    dibujar_c()
```

- Cada condición es controlada en orden. Si la primera es `False`, la siguiente es controlada y así sucesivamente.
- Si uno de ellos es `True`, la rama correspondiente se ejecuta y finaliza el bloque.
- Si más de una condición es verdadera, solo la primera se ejecuta.

Condiciones anidadas

- Un condicional puede ser también anidado dentro de otro. Podemos escribir el ejemplo tratado anteriormente de esta forma:

```
if x == y:
    print('x e y son iguales')
else:
    if x < y:
        print ('x es menor que y')
    else:
        print('x es mayor que y')
```

- El condicional exterior contiene dos ramas. La primera contiene una única sentencia simple. La segunda contiene otra sentencia **if** que tiene sus propias dos ramas.
- Esas dos ramas son también sentencias simples, sin embargo, podrían haber sido sentencias condicionales también.
- Puede parecernos que la identanción de las sentencias hace fácil de leer la estructura, pero los condicionales anidados se vuelven difíciles de leer muy rápido. Es una buena idea evitarlos lo más que podamos.

Condiciones anidadas (2)

- Los operadores lógicos nos pueden ser una solución para evitar los condicionales anidados. Por ejemplo, podemos reescribir el siguiente código usando un condicional simple:

```
if 0 < x:
    if x < 10:
        print('x es un número positivo de un sólo dígito')
```

- La instrucción **print** se ejecuta sólo si pasamos las dos condiciones, así que podemos obtener el mismo efecto utilizando un operador **and**:

```
if 0 < x and x < 10:
    print ('x es un número positivo de un sólo dígito')
```

- Para este tipo de condiciones, **Python** provee una opción más concisa:

```
if 0 < x < 10:
    print ('x es un número positivo de un sólo dígito')
```

Operador ternario

- El **operador ternario** es una herramienta muy potente que muchos lenguajes de programación tienen.
- En **Python** es un poco distinto a lo que sería en los lenguajes de la familia de C, pero el concepto es el mismo.
- Se trata de una cláusula `if, else` que se define en una sola línea utilizada que retorna un valor si se cumple una condición y otro en caso que no se cumpla.
- En vez de escribir:

```
a = 10, b = 10
if b != 0:
    c = a / b
else:
    c = -1
```

- Escribimos:

```
c = a/b if b != 0 else -1
```

Selección - Múltiples alternativas con match

- En este tipo de estructuras de control se nos permite ejecutar diferentes secciones de código dependiendo de una condición.
- La sentencia de control match/case toma un objeto y verifica si coincide con uno o más patrones. Por último lleva a cabo una acción si se encontraron coincidencias.
- Por ejemplo:

```
match variable:
    case 1:
      print("Seleccionó 1")
    case 2:
      print("Seleccionó 2")
    case _:
      print("No entró en ninguno")
```

- Donde si el valor de **variable** coincide con el de uno de los valores que siguen a la palabra reservada **case**, entonces, las sentencias dentro del bloque **case** se ejecutan.
- El símbolo **_** al final, es un comodín, se seleccionará en caso que no existan coincidencias con los valores expresados en las entradas **case** previas.

Múltiples alternativas con match (2)

- **Python** carecía hasta la versión 3.10 de una estructura de control del tipo **switch/case** como existe en la familia de lenguajes de C.
- Quedaban como alternativas entonces hacer bloques **if/elif** o utilizar diccionarios junto con expresiones lambda.
- En el caso de usar bloques **if/elif**, no todos los bloques tienen el mismo tiempo de acceso. Todas las condiciones van siendo evaluadas hasta que se cumple alguna de ellas y deriva el control a la sentencia siguiente. Imaginemos que tenemos 100 condiciones.

```
if variable == 1:
    print("1")
elif variable == 2:
    print("2")
# ... hasta 100
elif variable == 100:
    print("100")
else:
    print("x")
```

- El tiempo de ejecución será distinto si la variable es 1 o es 70 por ejemplo:
- Si es 1: Se evalúa el primer **if**, y como se cumple la condición se ejecuta y sale.
- Si es 70: Se va evaluando cada condición, hasta llegar al 70. Es decir, tienen que evaluarse 70 condiciones.

Bibliografía

- Óscar Ramírez Jiménez: ***"Python a fondo"*** 1era Edición. Ed. Marcombo S.L.. 2021.
- Allen Downey. ***"Think Python"***. 2Da Edición. Green Tea Press. 2015.
- Bill Lubanovic. ***"Introducing Python"***. 2Da Edición. O' Reilly. 2020.
- Eirc Matthes: ***"Python Crash Course"***. 1era Edición. Ed. No Starch Press. 2016.
- Zed A. Shaw: ***"Learn Python 3 the Hard Way"***. 1era Edición. Ed. Addison-Wesley. 2017.