NIVEL 2

INSTRUCCIONES CONDICIONALES

Universidad de los Andes

N2

¿CUÁNDO USAMOS INSTRUCCIONES CONDICIONALES?

- ✓ Cuando necesitamos dar una solución a un problema considerando distintos CASOS que se pueden presentar

- ✓ Dependiendo del CASO (que se expresa con una CONDICIÓN) se ejecuta una acción diferente

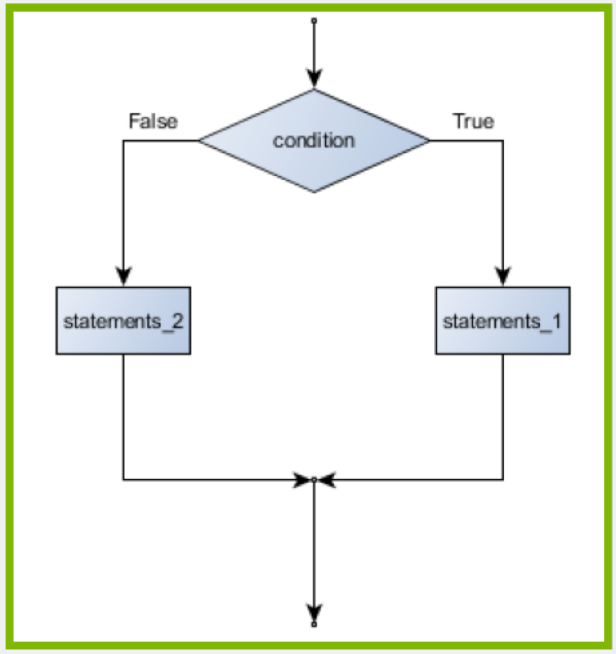
N2

INSTRUCCIÓN IF-ELSE

Esta es la «condición» del if.
Termina siempre con dos puntos :

```
if BOOLEAN EXPRESSION:
    STATEMENTS_1          # Executed if condition evaluates to True
else:
    STATEMENTS_2          # Executed if condition evaluates to False
```

Estas instrucciones son llamadas «bloques». No hay límite en la cantidad de instrucciones que componen un bloque, pero debe haber al menos una instrucción en cada bloque



N2

EJEMPLO: IF-ELSE

EjemploIfElse.py

```
1 x = int(input("Digite un número: "))
2
3 if x % 2 == 0:
4     print(x, " es par")
5     print("¿Sabía usted que 2 es el único número par que es primo?")
6 else:
7     print(x, " es impar")
8     print("¿Sabía usted que multiplicar dos números impares " +
9           "siempre da un resultado impar?")
10
```

Resultado de la ejecución

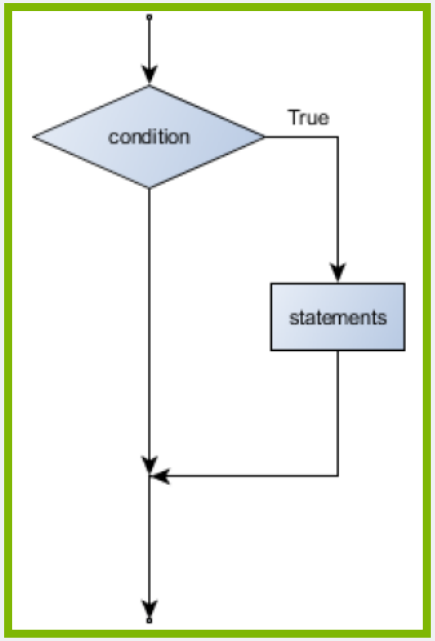
```
In [10]: runfile('C:/Users/Mhernandez/Desktop/IP/EjemploIfElse.py', wdir='C:/Users/

Digite un número: 10
10  es par
¿Sabía usted que 2 es el único número par que es primo?

In [11]: runfile('C:/Users/Mhernandez/Desktop/IP/EjemploIfElse.py', wdir='C:/Users/

Digite un número: 5
5  es impar
¿Sabía usted que multiplicar dos números impares siempre da un resultado impar?
```

N2

INSTRUCCIÓN IF SIN ELSE

```
if BOOLEAN EXPRESSION:
    STATEMENTS_1          # Executed if condition evaluates to True
```

Si la condición es verdadera, se ejecuta el bloque. De lo contrario, la ejecución del programa continúa en la instrucción después del if



EJEMPLO: IF SIN ELSE

EjemploIfSinElse.py

```
1  import math
2
3  x = int(input("Digite un número: "))
4
5  if x < 0:
6      print("El número negativo ", x, " no es válido aquí.")
7      y = x
8      x = 42
9      print("Decidí usar el número 42 en lugar de ", y)
10
11 print("La raíz cuadrada de ", x, "es", math.sqrt(x))
```

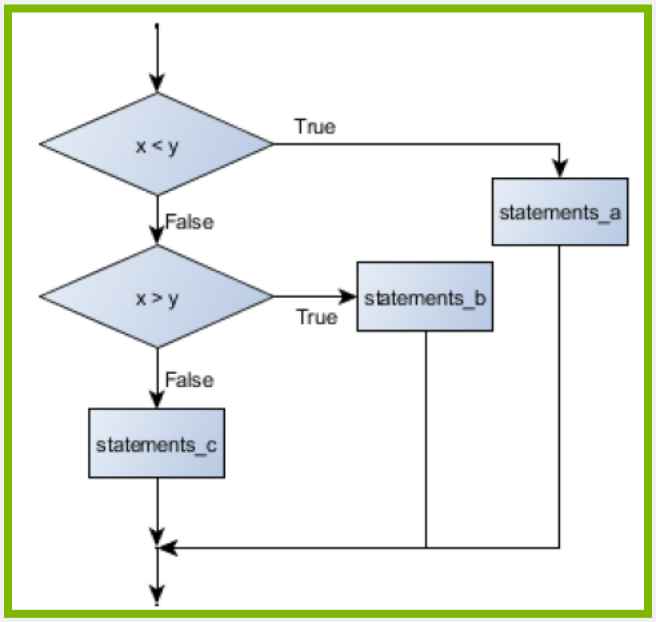
Resultado de la ejecución

Terminal 1/A

```
In [13]: runfile('C:/Users/Mhernandez/Desktop/I
IP/N2-C1')

Digite un número: 100
La raíz cuadrada de  100 es 10.0

In [14]: runfile('C:/Users/Mhernandez/Desktop/I
IP/N2-C1')

Digite un número: -25
El número negativo  -25  no es válido aquí.
Decidí usar el número 42 en lugar de  -25
La raíz cuadrada de  42 es 6.48074069840786
```

N2

INSTRUCCIONES CONDICIONALES EN CASCADA




```
if x < y:
    STATEMENTS_A
elif x > y:
    STATEMENTS_B
else:
    STATEMENTS_C
```

Se usan cuando hay más de dos posibilidades y necesitamos más de dos ramas de ejecución:

- ✓ elif es una abreviación de «else if»
- ✓ Sólo se ejecuta una rama (la de la primera condición que se satisfaga)
- ✓ No hay límite en la cantidad de «elif», pero debe haber un solo «else» al final

EJEMPLO: CONDICIONALES EN CASCADA

EjemploMenuIfCascada.py

```
1  def funcion_a()->None:
2      print ("Usted ha escogido la opción a del menú")
3
4  def funcion_b()->None:
5      print ("Usted ha escogido la opción b del menú")
6
7  def funcion_c()->None:
8      print ("Usted ha escogido la opción c del menu")
9
10 def funcion_d()->None:
11     print ("Usted ha escogido la opción d del menú")
12
13 #PROGRAMA PRINCIPAL
14
15 print ("Menu principal")
16 print ("Opción a")
17 print ("Opción b")
18 print ("Opción c")
19 print ("Opción d")
20
21 x = input("Seleccione su opción: ")
22
23 if x == "a":
24     funcion_a()
25 elif x == "b":
26     funcion_b()
27 elif x == "c":
28     funcion_c()
29 elif x == "d":
30     funcion_d()
31 else:
32     print("Selección inválida")
```

Resultado de la ejecución

Terminal 1/A

```
In [16]: runfile('C:/Users/Mhern
Desktop/IP/N2-C1')
Menu principal
Opción a
Opción b
Opción c
Opción d

Seleccione su opción: b
Usted ha escogido la opción b de

In [17]: runfile('C:/Users/Mhern
Desktop/IP/N2-C1')
Menu principal
Opción a
Opción b
Opción c
Opción d

Seleccione su opción: x
Selección inválida
```

N2

EJERCICIO

hands on

Escriba una función que reciba por parámetro un número entero y devuelva:

- ✓ -1 si el número es negativo
- ✓0 si el número es positivo pero menor a 1000
- ✓1 si el número es positivo y se encuentra entre 1000 y 10000
- ✓2 si el número es positivo y es mayor a 10000

Puedes verificar tus resultados usando la terminal presente en la actividad “Manos a la obra: Ejercicio de condicionales” en Brightspace

N2

UNA SOLUCIÓN CON VARIOS RETURNS

EjemploIfElIfVariosReturns.py*

```
1 def rango_numero(x: int)->int:
2     if x < 0:
3         return -1
4     elif x < 1000:
5         return 0
6     elif x < 10000:
7         return 1
8     else:
9         return 2
```

Resultado de la ejecución

Terminal 1/A

```
In [4]: runfile('C:/Users/Mhernand
Mhernandez/Desktop/IP/N2-C1')

In [5]: rango_numero(3)
Out[5]: 0

In [6]: rango_numero(-234)
Out[6]: -1

In [7]: rango_numero(1234)
Out[7]: 1

In [8]: rango_numero(15876)
Out[8]: 2
```

N2

OTRA SOLUCIÓN CON UN ÚNICO RETURN

- Se define una variable (en este caso respuesta)
- En cada condición se da un valor diferente a la variable
- Se retorna la variable al final

EjemploIfElIfUnSoloReturn.py

```
1  def rango_numero(x: int)->int:
2      if x < 0:
3          respuesta = -1
4      elif x < 1000:
5          respuesta = 0
6      elif x < 10000:
7          respuesta = 1
8      else:
9          respuesta = 2
10
11     return respuesta
```

Resultado de la ejecución

Terminal 1/A

```
In [4]: runfile('C:/Users/Mhernand
Mhernandez/Desktop/IP/N2-C1')

In [5]: rango_numero(3)
Out[5]: 0

In [6]: rango_numero(-234)
Out[6]: -1

In [7]: rango_numero(1234)
Out[7]: 1

In [8]: rango_numero(15876)
Out[8]: 2
```

N2

EJERCICIO

hands on

Escriba una función que reciba por parámetro cuatro números enteros y devuelva (retorne) el mayor de estos. Si hay dos o más iguales y mayores, retorna cualquiera de estos.

- ✓ ¿Es una sucesión de instrucciones condicionales en cascada?
- ✓ ¿Es una sucesión de instrucciones condicionales independientes?

Puedes verificar tus resultados usando la terminal presente en la actividad “Manos a la obra: Ejercicio de condicionales” en Brightspace

N2

DOS SOLUCIONES

Varios returns

EjemploMayorDeCuatroNumeros.py

```
1  def mayor_v1(a: int, b: int, c: int, d:int)->int:
2      if (a >= b) and (a >= c) and (a >= d):
3          return a
4      elif (b >= a) and (b >= c) and (b >= d):
5          return b
6      elif (c >= a) and (c >= b) and (c >= d):
7          return c
8      else:
9          return d
10
```

Un solo return

```
10
11 def mayor_v2(a: int, b: int, c: int, d:int)->int:
12     if (a >= b) and (a >= c) and (a >= d):
13         respuesta = a
14     elif (b >= a) and (b >= c) and (b >= d):
15         respuesta = b
16     elif (c >= a) and (c >= b) and (c >= d):
17         respuesta = c
18     else:
19         respuesta = d
20
21     return respuesta
```

Resultado de la ejecución

```
In [52]: mayor_v1(8,12,32,5)
Out[52]: 32

In [53]: mayor_v1(8,12,12,5)
Out[53]: 12

In [54]: mayor_v2(8,12,32,5)
Out[54]: 32

In [55]: mayor_v2(8,12,12,5)
Out[55]: 12
```

N2

UNA TERCERA SOLUCIÓN - ¡MÁS CORTA!

```
def mayor_optimo(a: int, b: int, c: int, d:int)->int:
    mayor = a
    if (b > mayor):
        mayor = b;
    if (c > mayor):
        mayor = c;
    if (d > mayor):
        mayor = d;

    return mayor
```

Estrategia de solución:

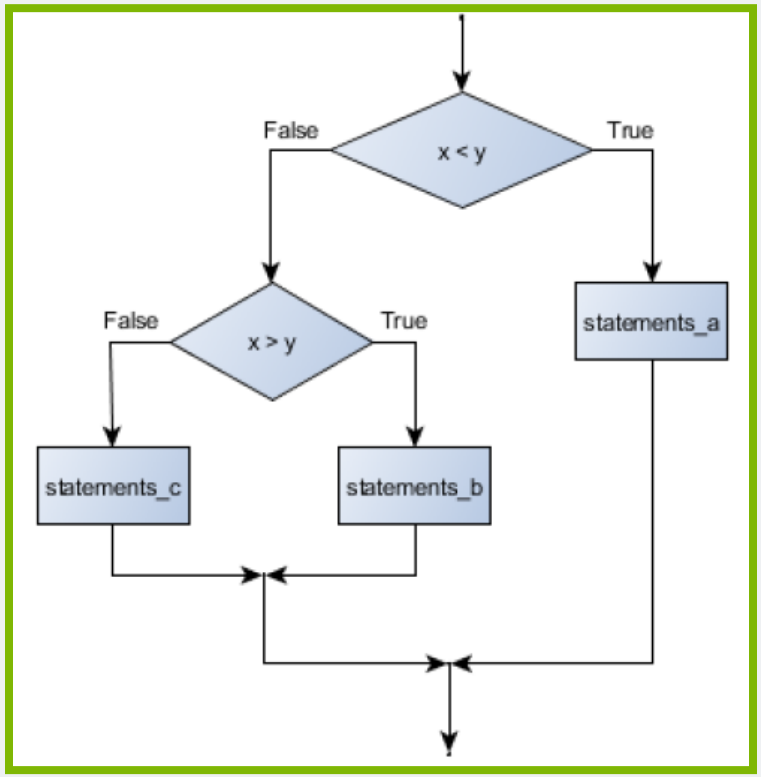
- ✓ Suponemos que el primero es el mayor y lo guardamos en una variable
- ✓ Si encontramos por el camino uno mayor que el mayor hasta el momento, remplazamos la variable con este nuevo valor

Resultado de la ejecución

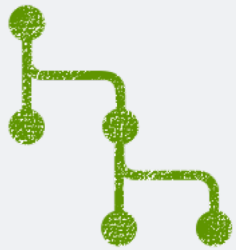
```
In [56]: mayor_optimo(8,12,32,5)
Out[56]: 32

In [57]: mayor_optimo(8,12,12,5)
Out[57]: 12
```

N2

INSTRUCCIONES CONDICIONALES ANIDADAS




```
if x < y:
    STATEMENTS_A
else:
    if x > y:
        STATEMENTS_B
    else:
        STATEMENTS_C
```



Las instrucciones condicionales pueden anidarse, es decir, aparecer unas «dentro» de otras

N2

EJEMPLO - TRES VERSIONES

- Escriba una función que reciba por parámetro un número entero y devuelva: verdadero si es un entero positivo de un dígito y falso de los contrario

EjemploEsPositivoUnSoloDigito.py

```
1  def es_positivo_de_un_solo_digito_version1(x: int)->bool:
2      if x > 0:
3          if x < 10:
4              return True
5          else:
6              return False
7      else:
8          return False
9
10 def es_positivo_de_un_solo_digito_version2(x: int)->bool:
11     if x > 0 and x < 10:
12             return True
13     else:
14         return False
15
16 def es_positivo_de_un_solo_digito_version3(x: int)->bool:
17     return x > 0 and x < 10
```

Versión 1: Condicionales anidados

Versión 2: Un solo condicional compuesto (and)

Versión 3: Retornando directamente la expresión condicional

Resultado de la ejecución

Terminal 1/A

```
In [7]: es_positivo_de_un_solo_digito_version1(-1)
Out[7]: False

In [8]: es_positivo_de_un_solo_digito_version2(1)
Out[8]: True

In [9]: es_positivo_de_un_solo_digito_version3(10)
Out[9]: False
```

N2

OPUESTOS LÓGICOS

Operador	Opuesto lógico
==	!=
!=	==
<	>=
<=	>
>	<=
>=	<

Cada uno de los 6 operadores relacionales tienen un opuesto lógico. Son útiles para expresar condiciones que requieren una negación ya que el operador de negación no siempre es evidente de usar

N2

OTRO EJEMPLO DE MÚLTIPLES VERSIONES DE LA MISMA FUNCIÓN

Versión 1: preguntando con una negación

EjemploPuedeTenerPase.py

```
1  def puede_tener_pase_version1(edad: int)->bool:
2      if not (edad >= 16):
3          return False
4      else:
5          return True
6
7  def puede_tener_pase_version2(edad: int)->bool:
8      if (edad < 16):
9          return False
10     else:
11         return True
```

Versión 2: preguntando con el opuesto lógico de la condición de la versión 1

N2

OTRO EJEMPLO DE MÚLTIPLES VERSIONES DE LA MISMA FUNCIÓN

Versión 3: usando una variable y retornándola al final

```
13 def puede_tener_pase_version3(edad: int)->bool:
14     puede = True
15     if (edad < 16):
16         puede = False
17     return puede
18
19 def puede_tener_pase_version4(edad: int)->bool:
20     puede = False
21     if (edad >= 16):
22         puede = True
23     return puede
24
25 def puede_tener_pase_version5(edad: int)->bool:
26     return (edad >= 16)
```

Versión 4: usando una variable con un valor por defecto

Versión 5: retornando directamente la expresión condicional