M3C5 Teoría

¿Qué es un condicional?

Los condicionales son instrucciones que ejecuta un bloque de código u otro, si cumplen determinados requisitos/condiciones.

La condición es una expresión lógica que es evaluada bajo las condiciones de verdadera o falsa. Si la condición es verdera la instrucción se cumplirá y se ejecutará. En caso contrario no, se ejecutará

Un condicional en Python es una estructura de control, que permite someter la ejecución del código a unas circunstancias/condiciones establecidas por el programador.

Las sentencias condicionales permiten controlar el flujo lógico de los programas, de una manera clara y compacta.

A continuación se trata las palabras reservadas **if/ else / elif / y las condicionales anidados**

Sentencia if

Si la expresión resulta ser VERDADERA/TRUE, entonces se ejecuta una vez el código.

En caso contrario y la expresión es FALSA/FALSE, entonces NO SE EJECUTA el código. Ejemplo:

```
a = 1

b = 2

if b > a:
print(" b es mayor que a")
```

Resultado:

```
b es mayor que a
```

La sintaxis general para la sentencia if básica :

- ✓ La palabra reservada if ,
- ✓ La siguiente parte es la condición. Esta puede evaluar si la declaración es verdadera o falsa. En Python estas son definidas por las palabras reservadas (True or False).
- ✓ Paréntesis (()) Los paréntesis son opcionales, pero ayudan a mejorar la legibilidad del código.
- ✓ Dos puntos : cuya función es separar la condición de la declaración de ejecución siguiente.
- ✓ Una nueva línea.
- ✓ Un nivel de indentación de **cuatro espacios**, que es una convención en Python.
- ✓ la estructura de la sentencia. Este es el código que será ejecutado, únicamente si la sentencia al ser evaluada es verdadera. En caso de múltiples líneas, hay que tener precaución que todas las líneas tengan el mismo nivel de indentación.

Sentencia if-else

Cuando la expresión if se evalúa como True, entonces ejecuta el código que le sigue. Pero si se evalúa como False, entonces ejecuta el código que sigue después de la sentencia else. Ejemplo:

```
a = 1
b = 2

if a < b:
print(" b es mayor que a")
else:
print("a es mayor que b")
```

En el ejemplo la línea de código que sigue la sentencia else, print("a es mayor que b"), nunca se ejecutará. La sentencia if anterior, a<b, se evaluó como True, por lo tanto es este el código ejecutado.

La sintaxis de una sentencia if-else es :

- ✓ Una sentencia else tiene como prerrequisito una sentencia if, siendo a la vez, parte de él.

- ✓ El siguiente código también necesita ser indentado **4 espacios** para definirlo como parte de la cláusula else.

- ✓ El código posterior a la sentencia condicional else se ejecuta *sí y solo sí* la primera parte, del condicional if se evalúa como Falsa. Si la sentencia en la condicional if se evalúa como Verdadera/True, será ese bloque de código el que se ejecutará y el código que sigue a else no se ejecutará nunca.

Sentencia if-elif-else

Si se necesita más de dos opciones. Es decir, si la primera condición es verdadera, realiza esto. Si esto no es verdadero, intenta esto otro, y si todas las condiciones fallan en ser verdaderas, entonces haz esto. Ejemplo

```
x = 1

if x > 10:
print(" x es mayor que 10")
elif x < 10:
print("x es menor que 10")
elif x < 20 :
print("x es menor que 20")
else:
print("x es igual a 10")
```

Resultado: x es menor que 10

En este ejemplo, if pone a prueba una condición específica, los bloques elif son dos alternativas, y el bloque final else es la solución final cuando todas las condiciones previas no se han cumplido.

Si todas las sentencias elif se evalúan como Falsas, entonces, y solo entonces se ejecutará el último bloque de código else.

Condicionales anidados

Son el resultado de introducir un condicional dentro de otro. Ejemplo:

```
x= 10
if x > 5:

    if x < 15:

        print("x está entre 5 y 15.")

    else:

        print("x es mayor que 15.")

else:

    print("x es menor o igual a 5.")
```

Las condiciones anidadas valoran cada requisito independientemente del resto.

https://aprendepython.es/core/controlflow/conditionals/

https://www.mclibre.org/consultar/python/lecciones/python-if-else.html

https://www.freecodecamp.org/espanol/news/sentencia-if-else-de-python-explicacion-de-las-sentencias-condiciones/

https://www.tokioschool.com/noticias/condicionales-python/

¿Cuáles son los diferentes tipos de bucles en Python? ¿Por qué son útiles?

Hay dos tipos de bucles en Python: bucle for y bucle while . Los bucles For se utilizan para iterar sobre una estructura de datos o secuencia de elementos, como una lista, cadena o diccionario, y ejecutar un bloque de código para cada elemento de la secuencia.

Los bucles son estructuras de control que nos permiten repetir un bloque de código varias veces.

Los **bucles for** y **while** funcionan diferente en función de cómo se itere el bloque de código.

El **bucle for** se aplica cuando el bloque a iterar tiene un **número definido**. El **bloque while** se aplica cuando la iteración **no tiene un número definido**; se repite mientras se cumpla una condición, mientras que el bucle **for** se utiliza cuando se conoce el número exacto de iteraciones que se deben realizar.

Bucle for

El bucle for se utiliza para recorrer los elementos de un objeto *iterable* (lista, tupla, conjunto, diccionario, …) y ejecutar un bloque de código. En cada paso de la iteración se tiene en cuenta a un único elemento del objeto iterable, sobre el cuál se pueden aplicar una serie de operaciones.

Bucles for en una lista o tupla

1- tenemos una lista de números y queremos mostrarlo. Ejemplos

```
nums = [4, 78, 9, 84]
for n in nums:
    print(n)
```

```
4
78
9
84
```

```
A = ["hola", 1, 65, "gracias", [2, 3]]
for value in A:
    print(value)
```

Resultado:

```
>
hola
1
65
gracias
[2, 3]
>
```

2- Iterar sobre dos listas del mismo tamaño en un solo bucle con la **función zip()**

```
A = ["a", "b", "c"]
B = ["a", "d", "e"]

for a, b in zip(A, B):
    print a, b, a == b
```

Resultado:

```
>
a a True
b d False
c e False
>
# True = Verdadero
# False = False
```

3- Iterarar sobre una lista y obtener el índice correspondiente con la **función enumerate() - enumerar**

```
A = ["esto", "es", "algo", "divertido"]

for indice,palabra in enumerate(A):
    print(indice, palabra)
```

Resultado:

```
>
0 esto
1 es
2 algo
3 divertido
>
```

Bucle for en diccionarios

Un caso especial de bucle for se da al recorrer los elementos de un diccionario. Dado que un diccionario está compuesto por pares clave/valor, hay distintas formas de iterar sobre ellas.

1 – Recorrer las claves del diccionario.

```
valores = {'A': 4, 'E': 3, 'I': 1, 'O': 0}
for k in valores:
    print(k)
```

```
A
E
I
O
```

Iterar sobre claves en un diccionario (también conocido como hashmap)

```
fruta_a_color = {"manzana": "#ff0000",
                 "lima": "#ffff00",
                 "naranja": "#ffa500"}

for key in fruta_a_color:
    print(key, fruta_a_color[key])
```

Resultado:

```
>
manzana #ff0000
lima #ffff00
naranja #ffa500
```

2 – Iterar sobre los valores del diccionario

```
valores = {'A': 4, 'E': 3, 'I': 1, 'O': 0}
for v in valores.values():
    print(v)
```

```
4
3
1
0
```

3 – Iterar a la vez sobre la clave y el valor de cada uno de los elementos del diccionario.

```
valores = {'A': 4, 'E': 3, 'I': 1, 'O': 0}
for k, v in valores.items():
    print('k=', k, ', v=', v)
```

```
k=A, v=4
k=E, v=3
k=I, v=1
k=O, v=0
```

Bucle for y la clase range

Implementación en Python de un bucle for basado en una secuencia numérica.

1- El constructor de esta clase, range(max), devuelve un iterable cuyos valores van desde 0 hasta max - 1.

```
for i in range(11):
    print(i)
```

```
0
1
2
3
...
```

2- muestra los números pares del 0 al 10 podríamos usando la función range del siguiente modo

```
for num in range(0, 11, 2):
    print(num)
```

```
0
2
4
6
8
10
```

Ejemplos:

```
for num in range(1, 10):
    print(num)
```

```
for num in range(1, 10, 2):
    print(num)
```

bucle for in <iterable>

Para implementar una función que devuelva la longitud de una lista.

```
def longitud(mi_lista):
    cont = 0
    for _ in mi_lista:
        cont += 1
    return cont
```

Modificando la iteración del bucle for: break, continue y else

1- Break

break se utiliza para finalizar y salir el bucle, por ejemplo, si se cumple alguna condición.

```
coleccion = [2, 4, 5, 7, 8, 9, 3, 4]
for e in coleccion:
    if e == 7:
        break
    print(e)
```

El código anterior mostrará los números 2, 4 y 5.

2- Continue

continue salta al siguiente paso de la iteración, ignorando todas las sentencias que le siguen y que forman parte del bucle. Mostrara los números pares.

```
coleccion = [2, 4, 5, 7, 8, 9, 3, 4]
for e in coleccion:
    if e % 2 != 0:
        continue
    print(e)
```

En este caso, el código anterior mostrará los números 2, 4, 8 y 4.

3- Else

Para poder hacer uso de else tendremos que utilizar una sentencia break y asi interrumpir el bucle for si una condición se cumple. Si el bucle for no es interrumpido la sentencia else se ejecutara

```
dias_semana = ['Lunes','Martes','Miercole','Jueves','Viernes']
hoy = 'Sabado'
for dia in dias_semana:
    if dia == hoy:
        print('Hoy es un dia laborable')
        break
    else:
        print('Hoy no es un dia laborable')
```

El código del bloque else se ejecutará siempre y cuando no se haya ejecutado la sentencia break dentro del bloque del for.

```
numeros = [1, 2, 4, 3, 5, 8, 6]
for n in numeros:
    if n == 3:
```

```
        break
else:
    print('No se encontró el número 3')
```

La secuencia numeros contiene al número 3, la instrucción print nunca se ejecutará.

Bucle while

ejecuta una porción de código mientras que la condición sea verdadera.

```
i = 0

while i < 5:
    print(i)
    i += 1
```

La variable i tiene valor 0 y se repitirá hasta cuatro veces ya que cuando llega a i=5 es falsa la premisa.

https://j2logo.com/bucle-for-en-python/

https://www.freecodecamp.org/espanol/news/bucles-for-en-python/

Python3 for loop documentation

https://tutorial.recursospython.com/bucles/

Recursos para aprender Python: guías, tutoriales y ejercicios

¿Qué es una lista por comprensión en Python?

La comprensión de listas es una construcción en Python que nos permite crear listas a partir de otras listas, tuplas y cualquier iterable. Nos permite escribir en forma más concisa. Es una funcionalidad que nos permite crear listas avanzadas en una misma línea de código

Podemos crear una lista con los cuadrados de los primeros 5 números de la siguiente forma

```
cuadrados = [i**2 for i in range(5)]
```

```
#[0, 1, 4, 9, 16]
```

De no existir, podríamos hacer lo mismo de la siguiente forma, pero necesitamos alguna que otra línea más de código.

```
cuadrados = []
for i in range(5):
cuadrados.append(i**2)
#[0, 1, 4, 9, 16]
```

Por un lado tenemos el for elemento in iterable, que itera un determinado iterable y "almacena" cada uno de los elementos en elemento. Por otro lado, tenemos la expresión, que es lo que será añadido a la lista en cada iteración.

La expresión puede ser una operación como hemos visto anteriormente i**2, pero también puede ser un valor constante. El siguiente ejemplo genera una lista de cinco unos.

```
unos = [1 for i in range(5)]
#[1, 1, 1, 1, 1]
```

https://www.tutorialesprogramacionya.com/pythonya/detalleconcepto.php?punto=94&codigo=95&inicio=90#:~:text=La%20comprensi%C3%B3n%20de%20listas%20es,y%20sin%20comprensi%C3%B3n%20de%20listas.

https://docs.python.org/es/dev/tutorial/datastructures.html

https://ellibrodepython.com/list-comprehension-python

¿Qué es un argumento en Python?

Los argumentos en funciones se refieren a los valores que se pasan a una función para que realice una tarea específica.
Se llama parámetro a la variable que se declara al definir la función (en donde se recibe el valor). Argumento al valor que es enviado a la función cuando se invoca.

Un argumento no será más que el valor que vamos a ingresar al momento de llamar a una función, y los parámetros serán variables definidas en la función misma que podrán almacenar los argumentos ingresados.

Ejemplo:

```
def suma(a, b, c):
    return a + b + c
```

En este caso hemos definido la función suma. Función que posee 3 parámetros (a, b y c). Estos parámetros no poseen valores por defecto, esto quiere decir que al invocar a la función, será necesario definir sus valores.

Ejemplos

```
# Argumentos por posición
suma(10, 20, 30)
60
```

En Python la asignación de argumentos se realiza por posición, esto quiere decir que el primer argumento es asignado al primer parámetro, el segundo argumento al segundo parámetro, el tercero al tercer parámetro. A esta asignación la conoceremos como positional argument.

Ejemplo

```
# Argumentos por nombre

suma(b=10, c=20, a=30)
60
```

En este caso, como podemos observar, asignamos valores ya no por posición, si no por nombre.

Lo interesante de la asignación por nombre, es que el orden no importa; solo basta con colocar el nombre del parámetro (Al cual queremos hacer la asignación), signo igual (=) y su correspondiente valor. A esta asignación la conoceremos como *keyword argument*.

NOTA: Una muy buena práctica en Python, es realizar la asignación por nombre sin espacio. variable=valor.

https://apuntes.de/python/manejo-de-argumentos-en-funciones-python-flexibilidad-y-adaptabilidad/#gsc.tab=0

https://docs.hektorprofe.net/python/programacion-de-funciones/argumentos-y-parametros/

https://aulasoftwarelibre.github.io/taller-de-python/Python_Avanzado/Argumentos/

https://platzi.com/discusiones/1764-python-cs/119642-una-pregunta-cual-es-la-diferencia-entre-argumento-y-parametro-ya-que-las-utilizo-muchas-veces-y-no-se-si-sean-lo-mismo/

https://es.stackoverflow.com/questions/446376/que-es-realmente-un-argumento-nombrado-en-python

https://www2.eii.uva.es/fund_inf/python/notebooks/08_Funciones/Funciones.html

https://pywombat.com/articles/argumentos-posiciones-nombres-python

¿Qué es una función Lambda en Python?

Las expresiones lambda se usan idealmente cuando necesitamos hacer algo simple y forma concisa, en lugar de nombrar formalmente la función.Las expresiones lambda también se conocen como funciones anónimas.

Las expresiones lambda en Python son una forma corta de declarar funciones pequeñas, necesita menos líneas de código y es más legible

Sintaxis de una función Lambda

lambda argumentos: expresión

Las funciones Lambda pueden tener cualquier número de argumentos, pero solo una expresión.

Ejemplo

```
# Función Lambda para calcular el cuadrado de un número
square = lambda x: x ** 2
print(square(3)) # Resultado: 9

# Funcion tradicional para calcular el cuadrado de un numero
def square1(num):
    return num ** 2
print(square(5)) # Resultado: 25
```

En el ejemplo de lambda anterior, lambda x: x ** 2 produce un objeto de función anónimo que se puede asociar con cualquier nombre.

https://www.freecodecamp.org/espanol/news/expresiones-lambda-en-python/

https://www.freecodecamp.org/espanol/news/funciones-lambda-en-python-ejemplos-de-sintaxis/

https://atareao.es/pyldora/las-maravillas-de-las-funciones-lambda-en-python/

https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/funciones-lambda-de-python

¿Qué es un paquete pip?

Pip es el sistema de gestión de paquetes utilizado para instalar y administrar paquetes/bibliotecas de software escritos en Python. Tiene una gran cantidad de paquetes disponibles,

Ejemplos

- ✓ Para instalar la biblioteca pandas y sus requisitos necesarios, escribiría lo **siguiente: pip install pandas[all]**
- ✓ Instalar un paquete: **pip install nombre_del_paquete**
- ✓ Actualizar un paquete: **pip install –upgrade nombre_del_paquete**
- ✓ Si queremos instalar una biblioteca como NumPy, **pip install numpy**
- ✓ Desinstalar un paquete: **pip uninstall nombre_del_paquete**
- ✓ Listar paquetes instalados: **pip list**

https://es.wikipedia.org/wiki/Pip_(administrador_de_paquetes)

https://abcxperts.com/que-es-pip/

https://apuntes.de/python/uso-de-pip-y-gestion-de-paquetes-en-python-administracion-de-dependencias/#gsc.tab=0

https://www.freecodecamp.org/espanol/news/como-usar-pip-install-en-python/

https://medium.com/@diego.coder/gesti%C3%B3n-de-dependencias-en-python-con-pip-e3977967889e