Universidad Tecnológica Nacional

Facultad Regional Avellaneda

Seminario de nivelación

Programación Inicial

Unidad 2 (asincrónica)

Índice de temas:

1. ***Condiciones y ejecución condicional:***
- Condicional.
- La sentencia if – Estructura condicional.
- Sentencias if – else anidadas.
- La sentencia elif.
- Análisis de muestras de código.

2. ***Operadores lógicos:***
- Operadores Lógicos.
- El operador and.
- El operador or.
- El operador not.

3. ***Bucles en tu código con While:***
- Bucles introducción.
- Un bucle infinito.
- Más ejemplos.
- Desafío.

4. ***Contadores y acumuladores:***
- Contadores.
- Acumuladores.

5. ***Máximos y mínimos:***
- Máximos y mínimos.
- Ejemplo máximo en Python.
- Ejemplo mínimo.

6. ***Validaciones:***
- Validaciones.
- Validaciones de rangos.

1. *Condiciones y ejecución condicional*

1.1 Condicional

Ya sabes cómo hacer preguntas a Python, pero aún no sabes cómo hacer un uso razonable de las respuestas. Se debe tener un mecanismo que le permita hacer algo si se cumple una condición, y no hacerlo si no se cumple.

Es como en la vida real: haces ciertas cosas o no cuando se cumple una condición específica, por ejemplo, sales a caminar si el clima es bueno, o te quedas en casa si está húmedo y frío.

Para tomar tales decisiones, Python ofrece una instrucción especial. Debido a su naturaleza y su aplicación, se denomina instrucción condicional (o sentencia condicional).

Existen varias variantes de la misma. Comenzaremos con la más simple, aumentando la dificultad lentamente.

La primera forma de una sentencia condicional, que puede ver a continuación, está escrita de manera muy informal (Pseudocódigo):

```
si (verdad o falso): realiza alguna acción si es verdad
```

Esta sentencia condicional consta de los siguientes elementos, estrictamente necesarios en este orden:

- La palabra clave reservada if**;**
- Uno o más espacios en blanco;
- Una **expresión** (una pregunta o una respuesta) cuyo valor se interpretar únicamente en términos de True y False)
- Unos dos puntos seguidos de una nueva línea;
- Una **instrucción con sangría** o un conjunto de instrucciones (se requiere absolutamente al menos una instrucción); la sangría se puede lograr de dos maneras: insertando un número particular de espacios (la recomendación es usar cuatro espacios de sangría), o usando el ***tabulador***; nota: si hay más de una instrucción en la parte con sangría, la sangría debe ser la misma en todas las líneas; aunque puede parecer lo mismo si se mezclan tabuladores con espacios, es importante que todas las sangrías sean exactamente iguales Python 3 no permite mezclar espacios y tabuladores para la sangría.

¿Cómo funciona esta sentencia?

- Si la expresión (verdad o falso) representa la verdad (es decir, su valor es igual a true), las sentencias con sangría se ejecutarán;
- Si la expresión (verdad o falso) representa falso, las sentencias con sangría se omitirán (ignorado), y la siguiente instrucción ejecutada será la siguiente al nivel de la sangría original.

En la vida real, a menudo expresamos un deseo:

si el clima es bueno, saldremos a caminar

1.2 La sentencia if - Estructura condicional

Las estructuras de decisión permiten elegir entre ejecutar una acción u otra, dependiendo de que la condición sea VERDADERA o FALSA para el valor que tienen asignadas las variables que aparecen en la misma cuando se ejecuta esta instrucción.

La palabra clave asociada a esta estructura es if

La forma más simple de un condicional es un if (del inglés sí) seguido de la condición a evaluar (evaluando el resultado que puede ser True o False) y dos puntos (:) al final de la línea. En las siguientes líneas e indentado, se escribe el código a ejecutar en caso de que dicha condición sea verdadera. Este bloque de instrucciones a ejecutar finaliza con la siguiente instrucción sin indentar.

```
if 25 >= 8: # Evaluar una expresión condicional
  print("puedo salir a caminar") # Ejecutar si la expresión condicional es verdadera
```

Puedes leerlo como sigue: si 25 es mayor o igual que 8, entonces imprime por pantalla "puedo salir a caminar"

Hemos dicho que las sentencias condicionales deben tener sangría. Esto crea una estructura muy legible, demostrando claramente todas las rutas de ejecución posibles en el código.

Otra forma de la sentencia alternativa if es la ejecución alternativa, en la que hay dos posibilidades y la condición determina cuál de las dos se ejecuta. La sintaxis en este caso es:

if condición:

sentencia 1

sentencia 2

else:

sentencia3

Si se debe ejecutar alguna acción cuando la condición es falsa, esta debe estar precedida por la palabra else:, a la altura del if al que se asocia. Éste es opcional, o sea que no tiene que colocarse si en el caso de que la condición sea falsa no hay que ejecutar ninguna sentencia.

```
if 7 >= 8: # Evaluar una expresión condicional
  print("puedo salir a caminar") # Ejecutar si la expresión condicional es verdadera
else:
  print("no puedo salir a caminar")# Ejecutar si la expresión condicional es Falsa
```

Mire la siguiente línea de código:

```
if True: # Evaluar una expresión condicional
  print("puedo salir a caminar") # Ejecutar si la expresión condicional es verdadera
else:
  print("no puedo salir a caminar")# Ejecutar si la expresión condicional es Falsa
```

Es importante recordar que también podemos setear de entrada la condición en este caso utilizamos True, siempre se ejecutara la primera sentencia.

IF EN PYTHON (8:40 min)**:**
https://www.youtube.com/watch?v=NxY0F_ScEFY&t=1s

1.3 Sentencias if-else anidadas

Ahora, analicemos dos casos especiales de la sentencia condicional.

Primero, considera el caso donde la instrucción colocada después del if es otro if.

Lee lo que hemos planeado ahora con una variable edad. Si es mayor de 18, ejecuta el código, pero a su vez dentro del primer IF hay otro condiciona que evalúa si es mayor a 22

Escribamos lo mismo en Python. Considera cuidadosamente el código siguiente:

```
edad = 25

if edad >= 18:
  if edad>22:
    print("Es mayor de edad y mayor a 22")
  else:
    print("Es mayor de edad pero menor a 22")
else:
  print("es menor de edad")
```

Aquí hay dos puntos importantes:

- Este uso de la sentencia if se conoce como anidamiento; recuerda que cada else se refiere al if que se encuentra en el mismo nivel de sangría; se necesita saber esto para determinar cómo se relacionan los *if*s y los *else*;
- Considera como la sangría mejora la legibilidad y hace que el código sea más fácil de entender y rastrear.

IF – ELSE EN PYTHON (5:03 min)**:**
https://www.youtube.com/watch?v=0WklTRogPzo

1.4 La sentencia elif

El segundo caso especial presenta otra nueva palabra clave de Python: **elif.** Como probablemente sospechas, es una forma más corta de else if.

elif se usa para verificar más de una condición, y para detener cuando se encuentra la primera sentencia verdadera.

Nuestro siguiente ejemplo se parece a la anidación, pero las similitudes son muy leves. Nuevamente, cambiaremos nuestros planes y los expresaremos de la siguiente manera: pidiendo una variable con la instrucción input

¿Has notado cuantas veces hemos usado las palabras *de lo contrario*? Esta es la etapa en la que la palabra clave reservada elif desempeña su función.

Escribamos el mismo escenario empleando Python:

```
valor = int(input("ingrese un numero del 1 al 7: "))

if valor==1:
  print("Es lunes")
elif valor==2:
  print("Es Martes")
elif valor==3:
  print("Es Miercoles")
else:
  print("No ha elegido ningun valor entre 1 y 7")
```

ELIF EN PYTHON (5:30 min):
https://www.youtube.com/watch?v=OpQzdLW5IgM

1.5 Análisis de muestras de código

Ahora te mostraremos algunos programas simples pero completos. No los explicaremos a detalle, porque consideramos que los comentarios (y los nombres de las variables) dentro del código son guías suficientes.

Todos los programas resuelven el mismo problema - encuentran el número mayor de una serie de números y lo imprimen.

Ejemplo 1:

Comenzaremos con el caso más simple - ¿cómo identificar el mayor de los dos números?:

```
# Se leen dos números
numero1 = int(input("Ingresa el primer número: "))
numero2 = int(input("Ingresa el segundo número: "))

# Elige el número más grande
if numero1 > numero2:
   numero_mayor = numero1
else:
   numero_mayor = numero1

# Imprime el resultado
print("El número más grande es:", numero_mayor)
```

El fragmento de código anterior debe estar claro - lee dos valores enteros, los compara y encuentra cuál es el más grande.

Ejemplo 2:

Ahora vamos a mostrarte un hecho intrigante. Python tiene una característica interesante - mira el código a continuación:

```
# Se leen dos números
numero1 = int(input("Ingresa el primer número: "))
numero2 = int(input("Ingresa el segundo número: "))

# Elige el número más grande
if numero1 > numero2: numero_mayor = numero1
else: numero_mayor = numero2

# Imprime el resultado
print("El número más grande es:", numero_mayor)
```

Nota: si alguna de las ramas de *if-elif-else* contiene una sola instrucción, puedes codificarla de forma más completa (no es necesario que aparezca una línea con sangría después de la palabra clave), pero solo continúa la línea después de los dos puntos).

Sin embargo, este estilo puede ser engañoso, y no lo vamos a usar en nuestros programas futuros, pero definitivamente vale la pena saber si quieres leer y entender los programas de otra persona.

No hay otras diferencias en el código.

Ejemplo 3:

Es hora de complicar el código - encontremos el mayor de los tres números. ¿Se ampliará el código? Un poco.

Suponemos que el primer valor es el más grande. Luego verificamos esta hipótesis con los dos valores restantes.

Observa el siguiente código:

```
# Se leen tres números
numero1 = int(input("Ingresa el primer número: "))
numero2 = int(input("Ingresa el segundo número: "))
numero3 = int(input("Ingresa el tercer número: "))

# Asumimos temporalmente que el primer número
# es el más grande.
# Lo verificaremos pronto.
numero_mayor = numero1

# Comprobamos si el segundo número es más grande que el mayor número actual
# y actualiza el número más grande si es necesario.
if numero2 > numero_mayor:
   numero_mayor = numero2

# Comprobamos si el tercer número es más grande que el mayor número actual
# y actualiza el número más grande si es necesario.
if numero3 > numero_mayor:
   numero_mayor = numero3

# Imprime el resultado.
print("El número más grande es:", numero_mayor)
```

ESTRUCTURA DE PROGRAMACIÓN CONDICIONAL (6:15 min)**:**
https://www.youtube.com/watch?v=JtmDA55J8KE

MATCH-CASE EN PYTHON. CONDICIONAL MÚLTIPLE (6:32 min)**:**
https://www.youtube.com/watch?v=Qstx0UArpH8

CONECTIVOS LÓGICOS EN PROGRAMACIÓN (8:30 min)**:**
https://www.youtube.com/watch?v=Frv69D45Ng0&t=4s

2. Operadores Lógicos

2.1 Operadores Lógicos

Lógica de computadoras

¿Te has dado cuenta de que las condiciones que hemos usado hasta ahora han sido muy simples, por no decir, bastante primitivas? Las condiciones que utilizamos en la vida real son mucho más complejas. Veamos este enunciado:

Si tenemos tiempo libre, y el clima es bueno, salimos a caminar.

Hemos utilizado la conjunción and **(y),** lo que significa que salir a caminar depende del cumplimiento simultáneo de estas dos condiciones. En el lenguaje de la lógica, tal conexión de condiciones se denomina **conjunción.** Y ahora otro ejemplo:

Si tu estás en el supermercado o la tienda de la esquina, uno de nosotros le comprará un regalo a juan.

La aparición de la palabra **or (o)** significa que la compra depende de al menos una de estas condiciones. En lógica, este compuesto se llama una **disyunción.**

Está claro que Python debe tener operadores para construir **conjunciones y disyunciones**. Sin ellos, el poder expresivo del lenguaje se debilitaría sustancialmente. Se llaman **operadores lógicos.**

2.2 El operador and

Un operador de conjunción lógica en Python es la palabra *and*. Es un operador binario con una prioridad inferior a la expresada por los operadores de comparación. Nos permite codificar condiciones complejas sin el uso de paréntesis como este:

```
vida = 3
energia = 70

print(vida > 0 and energia > 50)
```

El resultado proporcionado por el operador and se puede determinar sobre la base de la tabla de verdad.

Si consideramos la conjunción de A and B, el conjunto de valores posibles de argumentos y los valores correspondientes de conjunción se ve de la siguiente manera:

#	Argumento A	Argumento B	A and B
1	False	False	False
2	False	True	False
3	True	False	False
4	True	True	True

2.3 El operador or

Un operador de disyunción es la palabra or. Es un operador binario con una prioridad más baja que and (al igual que + en comparación con *). Su tabla de verdad es la siguiente:

```
vida = 0
energia = 70

print(vida > 0 or energia > 50)
```

El resultado proporcionado por el operador or se puede determinar sobre la base de la tabla de verdad.

Si consideramos la conjunción de A and B, el conjunto de valores posibles de argumentos y los valores correspondientes de conjunción se ve de la siguiente manera:

#	Argumento A	Argumento B	A and B
1	False	False	False
2	False	True	True
3	True	False	True
4	True	True	True

2.4 El operador not

Además, hay otro operador que se puede aplicar para condiciones de construcción. Es un operador unario que realiza una negación lógica. Su funcionamiento es simple: convierte la verdad en falso y lo falso en verdad.

```
soy_robot = True

print(soy_robot)

soy_robot = not soy_robot

print(soy_robot)

valorA = False

print(not valorA)
```

Este operador se escribe como la palabra not, y su prioridad es muy alta: igual que el unario + y -. Su tabla de verdad es simple:

#	Argumento A	not Argumento
1	False	True
2	True	False

OPERADORES LÓGICOS EN PYTHON (8:54 min)**:**
https://www.youtube.com/watch?v=C0w1PYYfYBA&t=1s

3. Bucles en tu código con while

3.1 Bucles introducción

¿Estás de acuerdo con la sentencia presentada a continuación?

```
#Mientras algo sea Verdad
  #mostra por pantalla infinitamente
```

Ten en cuenta que este registro también declara que, si algo es Falso, nada sucederá

En general, en Python, un bucle se puede representar de la siguiente manera:

```
while
  instruction
```

Si observas algunas similitudes con la instrucción if, está bien. De hecho, la diferencia sintáctica es solo una: usa la palabra while en lugar de la palabra if. La diferencia semántica es más importante: cuando se cumple la condición, if realiza sus sentencias sólo una vez; while repite la ejecución siempre que la condición se evalúe como True. Nota: todas las reglas relacionadas con sangría también se aplican aquí. Te mostraremos esto pronto.

Observa el algoritmo a continuación:

```
while conditional_expression:
  instruction_one
  instruction_two
  instruction_three
  :
  :
  instruction_n
```

Ahora, es importante recordar que: si deseas ejecutar más de una sentencia dentro de un while, debes (como con if) poner sangría a todas las instrucciones de la misma manera. una instrucción o conjunto de instrucciones ejecutadas dentro del while se llama el cuerpo del bucle. si la condición es False (igual a cero) tan pronto como se compruebe por primera vez, el cuerpo no se ejecuta ni una sola vez (ten en cuenta la analogía de no tener que hacer nada si no hay nada que hacer). el cuerpo debe poder cambiar el valor de la condición, porque si la condición es True al principio, el cuerpo podría funcionar continuamente hasta el infinito. Observa que hacer una cosa generalmente disminuye la cantidad de cosas por hacer.

3.2 Un bucle infinito

Un bucle infinito, también denominado bucle sin fin, es una secuencia de instrucciones en un programa que se repite indefinidamente (bucle sin fin).

Este es un ejemplo de un bucle que no puede finalizar su ejecución:

```
while True:
  print("Estoy atrapado dentro de un bucle.")
```

Este bucle imprimirá infinitamente "Estoy atrapado dentro de un bucle." en la pantalla.

Nota: Si deseas obtener la mejor experiencia de aprendizaje al ver cómo se comporta un bucle infinito, inicia IDLE, crea un nuevo archivo, copia y pega el código anterior, guarda tu archivo y ejecuta el programa. Lo que verás es la secuencia interminable de cadenas impresas de "Estoy atrapado dentro de un bucle." en la ventana de la consola de Python. Para finalizar tu programa, simplemente presiona *Ctrl-C* (o *Ctrl-Break* en algunas computadoras). Esto provocará la excepción KeyboardInterrupt y permitirá que tu programa salga del bucle. Hablaremos de ello más adelante en el curso.

Volvamos al bosquejo del algoritmo que te mostramos recientemente. Te mostraremos como usar este bucle recién aprendido para parar el bucle infinito

Analiza el programa cuidadosamente. Localiza donde comienza el bucle y descubre como se sale del cuerpo del bucle:

```
contador = 0

while contador < 3:
  print("debo aprender ciclos-este es el famoso contador", contador)
  contador = contador+1

print("termino el ciclo")
```

3.3 Más ejemplos

Veamos otro ejemplo utilizando el bucle while. Sigue los comentarios para descubrir la idea y la solución.

```
# Un programa que lee una secuencia de números
# y cuenta cuántos números son pares y cuántos son impares.
# El programa termina cuando se ingresa un cero.

numeros_pares = 0
numeros_impares = 0

# Lee el primer número.
numero = int(input("Introduce un número o escribe 0 para detener: "))

# 0 termina la ejecución.
while numero != 0:
  # Verificar si el número es impar.
  if numero % 2 == 1:
    # Incrementar el contador de números impares odd_numbers.
    numeros_impares += 1
  else:
    # Incrementar el contador de números pares even_numbers.
    numeros_pares += 1
  # Leer el siguiente número.
  numero = int(input("Introduce un número o escribe 0 para detener: "))

# Imprimir resultados.
print("Conteo de números impares:", numeros_impares)
print("Conteo de números pares:", numeros_pares)
```

3.4 Desafío

Adivina el número secreto

Escenario

Un mago junior ha elegido un número secreto. Lo ha escondido en una variable llamada secret_number. Quiere que todos los que ejecutan su programa jueguen el juego *Adivina el número secreto*, y adivina qué número ha elegido para ellos. ¡Quiénes no adivinen el número quedarán atrapados en un bucle sin fin para siempre! Desafortunadamente, él no sabe cómo completar el código.

Tu tarea es crear el código teniendo en cuenta los siguientes puntos:

- Pedirá al usuario que ingrese un número entero;
- Utilizará un bucle while;
- Comprobará si el número ingresado por el usuario es el mismo que el número escogido por el mago. Si el número elegido por el usuario es diferente al número secreto del mago, el usuario debería ver el mensaje "¡Ja, ja! ¡Estás atrapado en mi bucle!" y se le solicitará que ingrese un número nuevamente. Si el número ingresado por el usuario coincide con el número escogido por el mago, el número debe imprimirse en la pantalla, y el mago debe decir las siguientes palabras: "¡Bien hecho, Junior! Eres libre ahora."

¡El mago está contando contigo! No lo decepciones.

INFO EXTRA: Por cierto, observa la función print(). La forma en que lo hemos utilizado aquí se llama impresión multilínea. Puedes utilizar comillas triples para imprimir cadenas en varias líneas para facilitar la lectura del texto o crear un diseño especial basado en texto. Experimenta con ello.

```
secret_number = 777

print(
"""
+=================================+
| ¡Bienvenido a mi juego, Junior!|
| Introduce un número entero |
| y adivina qué número he |
| elegido para ti. |
|¿Cuál es el número secreto? |
+=================================+
""")
```

Deberás compartir tu Código en el Foro de consultas para que tus compañeros puedan probarlo.

CICLO WHILE EN PYTHON (4:34 min)**:**
https://www.youtube.com/watch?v=oDQJiRNzt98

4. Contadores y acumuladores.

4.1 Contadores

¿Qué es un contador en programación? ¿Qué es un acumulador en programación? Son conceptos de uso cotidiano que son el resultado de la suma de una serie de números. Los conceptos se pueden explicar con los siguientes ejemplos:

¿Qué es un contador en programación?

Al visitar el departamento de servicio al cliente en una empresa, los clientes para obtener un turno deben tomar un ticket. Un letrero electrónico indica el número del cliente que se está atendiendo, luego este número cambia incrementándose en 1 para anunciar el siguiente turno a ser atendido.

El ejemplo de uso práctico de un contador permite observar dos características:

- Siempre tienen un valor inicial
- El valor nuevo del contador es el resultado del valor anterior más una constante.

Al inicio del día, el contador de tickets debe ser inicializado, de preferencia con 0. Cuando un puesto de atención está listo para atención, el contador se incrementa en uno, se escucha una alerta y el cliente se puede acercar con el ticket del primer turno.

Las características descritas en forma algorítmica se escriben como:

```
contador = 0
contador = contador + 1
print(contador)
```

La expresión de la segunda línea se puede leer como: "valor nuevo" de contador es el resultado del "valor anterior" incrementando en 1.

Desde luego que los contadores pueden sumarse un valor diferente a 1, pero siempre será un valor constante, como cuando se cuenta de dos en dos.

La variable contadora también puede tener cambios de forma ascendente, o disminuir desde un valor inicial (decreciente).

Un ejemplo de contador decreciente se observa en cronometro del microondas para calentar alimentos. El valor inicial son los segundos que permanecerá encendido. El contador de tiempo disminuye en uno cada segundo y al llegar a 0 se apaga el microondas.

Ejemplo de contador con instrucciones en Python:

```
contador = 0
contador = contador + 1
print(contador)
contador = contador + 1
print(contador)
contador = contador + 1
print(contador)
```

4.2 Acumuladores

¿Qué es un acumulador en programación?

Un acumulador en programación es una versión ampliada de un contador. El acumulador tiene las mismas características que un contador excepto el valor de incremento que es un valor *variable*.

Por ejemplo, una cuenta de ahorros puede representarse en un algoritmo mediante un acumulador, pues quien ahorra no siempre lo hará con una cantidad fija en la cuenta: un día deposita 10, otro día deposita 30, otro deposita 5.

Con el ejemplo de ahorro, se puede determinar que en el acumulador no siempre se añade un valor positivo, pues cuando se hace un retiro, se puede interpretar como que el valor añadido es negativo.

Las características descritas para forma algorítmica se escriben como:

```
acumulador = 0
acumulador = acumulador + 20
acumulador = acumulador + 40
print(acumulador)
```

En el ejemplo anterior el valor se va acumulando dependiendo en este caso de los distintos valores que vamos agregando, ahora vamos a verlo pero pidiendo a el ingreso de datos para hacerlo más dinámico:

```
acumulador = 0
valorX = float(input("ingrese un valor: "))
acumulador = acumulador + valorX
valorX = float(input("ingrese un valor: "))
acumulador = acumulador + valorX
print("El valor acumulado es: ",acumulador)
```

Nota:

Es importante recordar que el ejemplo anterior lo estamos realizando para hacer solamente 2 ingresos.

¿Qué sucedería si necesito pedir 150 números? ¿este código sería efectivo? Debatan en los foros.

5. Máximos y mínimos

5.1 Máximos y mínimos:

Los números mayores y menores son conceptos fundamentales en matemáticas y además en el área de programación que nos permiten comparar cantidades o valores numéricos. Cuando comparamos dos números, determinamos cuál es más grande (mayor) y cuál es más pequeño (menor) en relación con el otro.

Para entender esto mejor, es importante recordar que los números están organizados en una línea numérica, donde cada número ocupa una posición específica según su magnitud. Por ejemplo, en la línea numérica:

0 - 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 10

Podemos ver claramente que el número 10 es mayor que el número 5, porque está más hacia la derecha en la línea numérica. Del mismo modo, el número 5 es mayor que el número 2, y así sucesivamente.

Cuando comparamos números, utilizamos los operadores relacionales para indicar la relación entre ellos:

- El símbolo ">" (mayor que) se utiliza para indicar que un número es mayor que otro. Por ejemplo, 7 > 3.
- El símbolo "<" (menor que) se utiliza para indicar que un número es menor que otro. Por ejemplo, 2 < 9.

Estos símbolos nos ayudan a expresar claramente las relaciones de orden entre los números y son fundamentales para resolver problemas, realizar comparaciones y tomar decisiones basadas en cantidades numéricas.

En resumen, entender los conceptos de número mayor y menor es esencial para desarrollar habilidades en programación y para la vida cotidiana, ya que nos permite comparar y entender las relaciones entre cantidades de manera clara y precisa.

5.2 Ejemplo máximo en Python

Veremos a continuación un ejemplo simple de como detectar por el momento el numero "Máximo" usando Python, ingresando cuatro números veamos la siguiente línea de código y analicemos unos instantes.

```
contador=0
max_numero = 0

while(contador<4):
  print("Ingrese un numero")
  num = int(input())

  if num > max_numero:
    max_numero = num

  contador+=1

print("ejecuciones: ",contador)
print("el valor maximo es: ", max_numero)
```

Si ejecutamos el código y hacemos varias pruebas, comprobaremos que por el momento funciona, recuerden que debemos probarlos con varias alternativas para testear nuestro código.

Una vez realizados cierta cantidad de pruebas, ¿Qué sucede si ingresamos todos números negativos? ´: -22, -25, -48, - 45

Para ello deberemos hacer una pequeña modificación a nuestro código, en principio asignándole el valor más chico que puede tener Python con la siguiente instrucción float('-inf'), que es menos infinito, el valor más chico.

```
contador=0
max_numero = float('-inf')

while(contador<4):
  print("Ingrese un numero")
  num = int(input())

  if num > max_numero:
    max_numero = num

  contador+=1

print("ejecuciones: ",contador)
print("el valor maximo es: ", max_numero)
```

5.3 Ejemplo Mínimo

Veremos a continuación un ejemplo simple de como detectar por el momento el numero "Mínimo" usando Python, ingresando cuatro números veamos la siguiente línea de código y analicemos unos instantes.

```
contador=0
min_numero = 0

while(contador<4):
   print("Ingrese un numero")
   num = int(input())

   if num < min_numero:
      min_numero = num

   contador+=1

print("ejecuciones: ",contador)
print("el valor minimo es: ", min_numero)
```

Si ejecutamos el código y hacemos varias pruebas, comprobaremos que por el momento funciona, recuerden que debemos probarlos con varias alternativas para testear nuestro código.

Una vez realizados cierta cantidad de pruebas, ¿Qué sucede si ingresamos todos números negativos? ´: -22, -25, -48, - 45

Para ello deberemos hacer una pequeña modificación a nuestro código, en principio asignándole el valor más chico que puede tener Python con la siguiente instrucción float('-inf'), que es menos infinito, el valor más chico.

```
contador=0
min_numero = float('inf')

while(contador<4):
   print("Ingrese un numero")
   num = int(input())

   if num < min_numero:
      min_numero = num

   contador+=1

print("ejecuciones: ",contador)
print("el valor min es: ", min_numero)
```

¡Desafío Máximos y Mínimos!

Ahora el siguiente desafío consiste en hacer que el programa que tuvimos de prueba muestre tanto el valor máximo ingresado como el minio.

Ahora... si lograste realizarlo utilizando el código de muestra hasta el momento, ¿qué sucede si ingresamos números consecutivos como 1,2,3,4?

6. Validaciones

6.1 Validaciones

Cuando le pedimos un dato al usuario, es posible que ese dato se tenga que validar, esto significa darle un valor legal a una variable, es decir que la variable guarde en si un valor permitido.

vamos a ver 3 pequeños casos puntuales con sus ejemplos pertinentes.

El siguiente ejemplo muestra cómo realizar una validación para un único valor

```
clave = input("ingrese la clave: ")
clave = int(clave)

while clave != 666:
  clave = int(input("ingrese nuevamente la clave: "))
#si ingresamos bien la clave continua...
print("Bienvenidos al Curso de Nivelación")
```

¡Nota!

Para hacer una validación recomendamos preguntar por lo que no queremos que se cumpla, en el ejemplo anterior el ejemplo claro seria:

clave != 666:

6.2 Validación de rangos.

El siguiente ejemplo muestra cómo realizar una validación para un rango de valores posibles, analice el código y debata con sus compañeros.

```
nota = input("ingrese la nota")
nota = int(nota)

while nota < 1 or nota > 10:
  nota = input("ingrese nuevamente la nota")
```