Tema 1:
Python 2

Índice

1. Operadores lógicos 2
1.1. Operadores a nivel de bits 2
2. Operadores de comparación 3
3. Estructuras de control 3
3.1. Sentencias condicional if 4
3.2. if-else 4
3.3. if-elif-else 4
3.4. Consideraciones a tener en cuenta en bucles if 4
4. Funciones 6
4.1. Parámetros 6
4.2. return 7
4.3. None 7
4.4. Docstring 8
5. Bucles 8
5.1. while 8
5.2. for 8
5.3. Función range() 9
5.4. Bucles anidados 9
5.5. La rama else de los bucles 10
6. Break, continue y pass 10
6.1. break 10
6.2. Continue 10
6.3. Pass 10

1. Operadores lógicos

Los operadores lógicos tienen dos posibles valores: verdadero y falso. Sin embargo debido a que en ciertas ocasiones el valor de un elemento no está definido o no es conocido estos dos valores no son suficientes. Para solucionar este problema se hace uso de la lógica trivaluada en la cual a los valores verdadero y falso se le añade el valor desconocido/nulo (NULL).

En este apartado se verán los operadores lógicos binarios Y y O además del operador unario No.

Las tablas de verdad de estos operadores son:

A	B	A and B	A or B
False	False	False	False
False	True	False	True
True	False	False	True
True	True	True	True

A	not A
False	True
True	False

Donde:

- Y → representado por and: su resultado es verdadero cuando los dos operandos son verdaderos. Falso si por lo menos uno de ellos es falso (falso y otro valor cualquiera es falso).
- O → representado por or: es falso si los dos operadores son falsos. Verdadero si por lo menos uno de ellos es verdadero.
- No → representado por not: si es verdadero el resultado es falso y viceversa.

```
ciudad = "Ourense"
edad = 20
print(not (ciudad == "Vigo") and edad >= 18)
```
```
True
```

Estos operadores de pueden asociar pero hay que tener en cuenta el orden de prioridad. De más prioritario a menos:

1. +, - unario
2. **
3. *, /, //, %
4. +, - binario
5. <, <=, >, >=, ==, !=
6. not
7. and
8. or

1.1. Operadores a nivel de bits

Los operadores a nivel de bits actúan a nivel de bits y permiten desplazar esto.

- a >> b desplaza a b bits a la derecha
- a << b desplaza a b bits a la izquierda

```
print(8<<1)
```
```
16
```

```
print(8>>1)
```
```
4
```

2. Operadores de comparación

Nos permiten comparar dos expresiones

Tenemos los siguientes operadores:

Operador	Nombre operador	Uso
==	Igualdad	Devuelve verdadero si el resultado de las expresiones es igual. Falso si son diferentes.
!=	Desigualdad	Devuelve verdadero si el resultado de las expresiones es diferente. Falso si son iguales.
<	Menor que	Devuelve verdadero si la primera expresión es menor que la segunda. Falso si es mayor o igual.
<=	Menor o igual	Devuelve verdadero si la primera expresión es menor o igual que la segunda. Falso si es mayor.
>	Mayor	Devuelve verdadero si la primera expresión es mayor que la segunda. Falso si es menor o igual.
>=	Mayor o igual	Devuelve verdadero si la primera expresión es mayor o igual que la segunda. Falso si es menor.

Ejemplos:

```
numero = 2
print(numero == 2)
```
```
True
```

```
numero = int(input("Escribe un número:"))
print(numero == 2)
```

```
print(2 == "2")
```
```
False
```

```
print("100" > "50")
```
```
False
```

3. Estructuras de control

Hasta ahora hemos visto como ejecutar instrucciones de forma secuencial, las sentencias se ejecutan una a continuación de otra, pero, en la mayoría de los programas, es necesario tomar decisiones para ejecutar un código u otro.

En la ejecución del código, se van a ir planteando preguntas que solo tienen dos respuestas posibles:

- Sí es verdadero → True
- Si es falso → False

No existe ambigüedad posible.

Python tiene la característica de que la forma de indicar si un elemento está o no dentro de una estructura es mediante la identación (se recomiendan 4 espacios).

3.1. Sentencias condicional if

Mecanismo que permita ejecutar instrucciones si se cumple una condición, y no hacerlo si no se cumple.

```
if condicion:
    # Código a ejecutar si se cumple la condición
```

Si se cumple la condición todo el código a ejecutar debe aparecer identado. Si el código a ejecutar consta de varias sentencias, todas ellas deben estar identadas.

La condición debe ser una expresión cuyo valor se interpretará únicamente en términos de True y False.

```
edad= int(input(("Indica tu edad: "))
if edad>=18:
    print ("Puedes acceder")
    print ("Adelante!")

print("Esta línea se ejecuta siempre")  # No está dentro del if
```

3.2. if-else

La cláusula else nos permite indicar qué sentencias se ejecutarán cuando **no** se cumple la condición.

```
if condicion:
    # Código a ejecutar si se cumple la condición
else:
    # Código a ejecutar si NO se cumple la condición
```

```
edad = int(input("Indica tu edad: "))
if edad >= 18:
    print("Eres mayor de edad")
else:
    print("Eres menor de edad")

print("La edad solo es un número")
```

3.3. if-elif-else

Sirve para indicar alternativas. Es la forma abreviada de else if. Permite indicar una nueva condición si no se cumplieron las anteriores.

```
n= input("Introduce un número ")
numero=int(n)
if numero< 0:
    print ('Negativo')
elif numero> 0:
    print ('Positivo')
else:
    print ('Cero')
```

3.4. Consideraciones a tener en cuenta en bucles if

- Como mucho se ejecuta una de las ramas.
- No se puede usar else si antes no se ha usado un if.
- else siempre es la última rama del if, independientemente de si has usado elif o no.
- else es una parte opcional y puede omitirse.
- Si no hay una rama else, es posible que no se ejecute ninguna de las opciones disponibles.
- Si hay una rama else, siempre se ejecuta una, y solo una, de las ramas.

Ejercicios

1. Implementa un programa en Python que pida por teclado un número entero y que indique si es par o impar.
2. Implementa un programa en Python que pida dos números por teclado y que muestre por pantalla al menor de los dos o si son iguales.
3. Realizar un programa que pida dos números al usuario en sendas variables y luego intercambie el valor de dichas variables. Finalmente mostrará en pantalla ambas variables.
4. Implementa un programa en Python que pida un número por teclado y que muestre por pantalla un mensaje indicando si es o no negativo. El 0 no es negativo.
5. Implementa un programa en Python que pida tres números por teclado y muestre por pantalla el mayor de los tres.
6. Queremos implementar un programa que permite convertir temperaturas. Al usuario se le piden los grados Celsius y se le pregunta si los quiere convertir a Farenheit o a Kelvin. Las formulas de conversión son:

   ^{0}Kelvin = ^{0}Celsius + 273

   ^{0}Farenheit = ^{0}Celsius*1.8 + 32
7. Implementa un programa en Python para mostrar por pantalla la calificación de una nota a partir de la puntuación introducida por teclado:
   - [0,5) Suspenso
   - [5,6) Aprobado
   - [6,7) Bien
   - [7,9) Notable
   - [9,10] Sobresaliente

4. Funciones

Las funciones permiten crear pequeños fragmentos de código, que realizan una acción, a los que se les da un nombre y que pueden ser ejecutadas en cualquier parte del código.

Con ello se busca:

- Disminuir la complejidad del programa. Usando la máxima de divide y vencerás: un programa complejo se puede dividir en partes más sencillas.
- Reutilización de código.
- Facilitar el mantenimiento. Un código que se podría escribir en varias partes del programa se escribe en una única parte por lo que si hay que modificar solo se modificaría en un único lugar.

El formato de su definición es la siguiente:

```
def nombre_funcion(parametro1, parametro2, …):
    cuerpo de la función
    return dato  # Esto es opcional
```

Donde:

- Se inicia la definición mediante la palabra reservada **def**
- Un nombre con las mismas reglas que las variables.
- Una lista de parámetros entre paréntesis o unos paréntesis vacios si no tiene parámetros.
- Dos puntos.
- El cuerpo de la función con al menos una línea. El cuerpo de la función se define mediante la identación.
- Si el método devuelve un valor tendremos la palabra reservada return. Lo vemos más adelante.

Según su tipo tenemos los siguientes tipos de funciones:

- Funciones integradas en Python, por ejemplo, print().
- Funciones de los módulos preinstalados, por ejemplo, math, time, etc.
- Funciones definidas por el programador.
- Métodos de las clases (POO).

4.1. Parámetros

A una función se le pueden pasar parámetros.

Donde:

- No se especifica su tipo (a partir de la versión 3.5 se pueden especificar pero no es obligatorio).
- Solo son visibles en el cuerpo de la función.
- Sus valores pueden ser literales, variables, funciones, expresiones, …
- En una invocación los argumentos se pueden especificar de forma posicional. El primer argumento de la invocación se corresponde con el primero parámetro de la función, el segundo con el segundo, …

```
def presentacion(nombre, apellido):
    print("Hola, me llamo", nombre, apellido)

presentacion("Lorenzo", "Iglesias")
```

```
Hola, me llamo Lorenzo Iglesias
```

- Los argumentos se pueden especificar también con una palabra clave (un nombre):

```
def presentacion(nombre, apellido):
    print("Hola, me llamo", nombre, apellido)

presentacion(nombre="Lorenzo", apellido="Iglesias")
```
```
Hola, me llamo Lorenzo Iglesias
```
```
presentacion(apellido="Freire", nombre="Manuela")
```
```
Hola, me llamo Manuela Freire
```

- Se pueden combinar ambos tipos de argumentos pero se han de colocar, de forma obligatoria, todos los argumentos posicionales antes que los de palabra clave.

```
presentacion("Lorenzo", apellido ="Iglesias")
```
```
Hola, me llamo Lorenzo Iglesias
```

- Se pueden definir valores predefinidos para los parámetros. Este es el valor que tomará el parámetro si ese parámetro no se especifica en la invocación.

```
def presentacion(nombre, apellido1="Graiño", apellido2=""):
    print("Hola, me llamo", nombre, apellido1, apellido2)

presentacion("María", apellido2="Senra")
```
```
Hola, me llamo María Graiño Senra
```

4.2. return

return provoca la terminación inmediata de la función.

Puede:

- No devolver nada.

```
return
```

- Devolver un valor. Este valor puede ser un literal, una expresión, …

```
def area_triangulo(base, altura):
    return base * altura / 2

area = area_triangulo(10, 20) # El resultado puede guardarse
```

- Devolver varios valores.

```
def area_perimetro_rectangulo(base, altura):
    area = base * altura
    perimetro = 2 * (base + altura)
    return area, perímetro

a, p = area_perimetro_rectangulo(3, 5)
```

La asignación de la devolución es posicional. El primer valor devuelvo se asigna a la primera variable, el segundo a la segunda variable, …

4.3. None

Es una palabra clave reservada e indica la falta de valor.

Solo existen dos tipos de circunstancias en las que None se puede usar de manera segura:

- Cuando se le asigna a una variable (o se devuelve como el resultado de una función).
- Cuando se compara con una variable para diagnosticar su estado interno.

```
value = None
if value is None: # if value == None:
    print("Lo siento, no contienes ningún valor")
```

Si una función no devuelve un valor utilizando return, se asume que devuelve implícitamente None.

4.4. Docstring

Al definir una función propia utilizaremos el docstring para documentar la función. Este texto será el que aparezca si algún usuario hace un help sobre el nombre de nuestra función.

El docstring se escribe después de la definición del nombre de la función entre comillas triples.

Ejemplo:

```
def area_triangulo(base, altura):
    ''' base: base del triangulo
        altura: altura del triangulo
        devuelve el área de un triangulo al pasarle su base y su alura
        '''
    return base * altura / 2
```

5. Bucles

Un bucle, o ciclo, es un conjunto de instrucciones que se repiten un número de veces o mientras se sigue cumpliendo una condición.

5.1. while

Las instrucciones dentro del while se ejecutan mientras la condición sea verdadera.

```
while condicion:
    # Sentencias a ejecutar mientras se cumpla la condición
```

Ejemplo:

```
edad=0
while edad <18:
    edad = input("Introduce edad> ")
    edad = int(edad)
print("Ahora puedes pasar")
```

Si en la primera comprobación la condición del while es falsa el bucle no se ejecuta.

Se permite el uso de break y continue.

5.2. for

En bucle for permite la ejecución de un conjunto de instrucciones un número determinado de veces. Es similar al foreach de otros lenguajes;

```
for i in 1, 2, 3, 6: # los valores no tienen por qué ser consecutivos
    print("El valor de i es", i)
```

```
El valor de i es 1
El valor de i es 2
El valor de i es 3
El valor de i es 6
```

Donde:

- La palabra reservada **for** abre el bucle
- La variable después de la palabra reservada for es la variable de control del bucle; cuenta automáticamente las iteraciones del bucle.
- La palabra reservada **in** introduce el rango de valores posibles que se asignan a la variable de control.
- En cada iteración, la variable de control tomará uno de los valores del rango.

Para generar los números de la secuencia de forma automática usamos la función range.

```
for i in range(4):
    print("El valor de i es", i)
```

```
El valor de i es 0
El valor de i es 1
El valor de i es 2
El valor de i es 3
```

El bucle for también se usa para recorrer secuencias de datos, por ejemplo, cadenas.

```
palabra =  "Hola mundo"
for letra in palabra:
    print(letra, end="-")
```

```
H-o-l-a- -m-u-n-d-o-
```

En este bucle se van obteniendo carácter a carácter cada uno de los caracteres de la palabra.

5.3. Función range()

Es una función que produce una secuencia de enteros. Donde:

- Cuando solo tiene un argumento, genera números enteros desde 0 al valor de ese argumento menos 1.

```
for i in range(5):  # range(fin)
    print(i, end="")
```

```
01234
```

- Cuando tiene dos argumentos, genera números enteros desde el valor del primer argumento al valor del segundo argumento menos 1.

```
for i in range(2,7):  # range(inicio, fin)
    print(i, end="")
```

```
23456
```

- Cuando tiene 3 argumentos, el tercero indica el incremento que se produce cada vez.

```
for i in range(1, 7, 2 ):  # range(inicio, fin, incremento)
    print(i, end="")
```

```
135
```

- Para realizar un for decreciente se usa un rango invertido y un incremento negativo.

```
for i in range(9, 4, -1 ):
    print(i, end="")
```

```
98765
```

5.4. Bucles anidados

Dentro de un bucle puede encontrarse otro bucle, de esta manera, para cada ejecución del bucle externo, se ejecutará por completo el bucle interno.

El siguiente ejemplo incrementa en cada línea el número de asteriscos:

```
for i in range(1,10):
    for j in range(1,i):
        print("*", end="")
    print()
```

El anterior es un código de ejemplo. Se podría simplificar de la siguiente forma:

```
for i in range(1,10):
    print(i*"*")
```

Ejemplo de un bucle decreciente:

```
for i in range(10, 1, -1):
    for j in range(1,i):
        print("*", end="")
    print()
```

O de iteración sobre cadenas:

```
for piso in range(1,5):
    for letra in "ABCD":
        print(piso, letra, sep = "º", end=" ")
    print()
```

5.5. La rama else de los bucles

La rama else del bucle siempre se ejecuta una vez, independientemente de si el bucle ha entrado o no (porque no se ha cumplido la condición al principio) en su cuerpo.

La rama else no se ejecuta si se ha salido del bucle con una sentencia break.

Por ejemplo:

```
for x in range(1,10):
    for y in range(1,x):
        print("*", end="")
    print()
else:
    print("fin")
```

6. Break, continue y pass

Se utilizan tanto para los bucles while como for.

6.1. break

La instrucción break se utiliza para salir de un bucle antes de que este termine.

```
for i in range(10):
    print(i, end=" ")
    if i == 5:
        break  # Salimos del bucle
```

```
1 2 3 4 5
```

La alternativa sin utilizar break, es usando una bandera y cambiando a while.

```
fin = False
contador = 0
while (not fin and contador<10):
    print(contador, end=" ")
    if contador == 5:
        fin=True  # Salimos del bucle
    contador+=1
```

```
0 1 2 3 4 5
```

6.2. Continue

Permite abandonar el flujo normal del buche y fuerza a comenzar una nueva iteración

```
for i in range(10):
    if i == 5:
        continue # Saltamos esta iteración, no imprime el 5
    print(i, end=" ")
```

```
0 1 2 3 4 6 7 8 9
```

6.3. Pass

Se utiliza como una declaración de relleno o de lugar; no tiene efecto. Por ejemplo para definir una función con el cuerpo vacio.

Ejercicios

Cada uno de estos ejercicios se ha de implementar en una función diferente.

8. Realiza un programa con un bucle for que muestre los números pares entre 1 y un valor pasado como parámetro..
9. Crea un programa en Python que muestre los números desde un número mayor hasta otro menor introducidos por teclado. Comprueba que el primer número es mayor que el segundo, y si no lo es, muestra un mensaje y finaliza el programa.
10. Crea un programa que verifique que la contraseña introducida sea correcta. Para ello pide al usuario una contraseña y almacénala. Luego, utilizando un bucle, ve pidiendo la contraseña hasta que coincida con la almacenada. Muestra el número de intentos realizados.
11. Implementa un programa que muestre continuamente este menú hasta que el usuario elija la opción de salir:
    1. Saludar
    2. Despedir
    3. Salir

    Si el usuario elige "Saludar", el programa escribirá "Hola"; si elige "Despedir", escribirá "Adiós". En ambos casos el programa continua.

    Si elige "Salir", se detendrá la ejecución del programa.
12. Realizar un programa que calcule la media de una serie de números introducidos por teclado. Cuando el usuario introduzca un 0 se ha de mostrar resultado por pantalla y permitir que el programa continúe para calcular una nueva media.
13. Queremos escribir un programa que permita encontrar el primer valor donde la suma de una secuencia supere una cantidad. Por ejemplo la suma 1+2+3+...+N supera un parámetro. Si el parámetro es 2000 la solución será 63.
14. Realizar un programa que calcule el factorial de un número introducido por el usuario
    - Con un bucle for.
    - Con un bucle while

    Nota: El factorial de un número es el número multiplicado por todos sus anteriores.

    Como por ejemplo el factorial de 5: 5!= 5*4*3*2*1.
15. Realizar un programa que saque los siguientes datos por pantalla (usa bucles for y haz en un único programa todos los apartados).

    a) Los números del 1 al 50.

    b) En orden inversa del 50 al 1

    c) Los números pares menores que 20 por orden creciente (2, 4, 6, ... 16, 18)

    d) Los números impares entre 90 y el 130 por orden creciente

    e) Los múltiplos de 5 entre el 70 y el 25 por orden decreciente (70, 65, ... 30,25).
16. Crea un programa que pida al usuario una frase y le diga cuántas vocales tiene esa frase.
17. Vamos a crear una versión del juego Adivina un número en el cual al jugador ha

de encontrar un número entre 1 y 100. Dónde.

- El jugador que inicia el juego tiene que introducir un número entre 1 y 100. Mientras el número no esté en ese rango se siguen pidiendo números. Tras introducir un valor correcto debe borrar la pantalla imprimiendo 50 líneas en blanco.
- El jugador tienen 5 intentos para encontrar el número. Si el número introducido:
  - Es menor: mostrará el mensaje es mayor.
  - Es mayor: mostrará el mensaje es menor.
  - Si es igual: mostrará el mensaje bien hecho y termina la partida.
- Si el número de intentos es 0 la partida termina.
- Siempre al terminar la partida da la opción de salir del juego o comenzar otra partida.
- Si el número introducido es igual mostrara el mensaje: bien hecho. Has necesitado x intentos. Donde x es el número de intentos realizados.

18. Vamos a modificar el ejercicio anterior para que en vez de preguntar el número a adivinar usando la función random.randrange(numMax) lo genere de forma automática. Esta función genera un número aleatorio entre 0 y numMax-1.
19. Crea los programa para que muestren las siguientes pirámides. Los programas siguientes tienen que ser genéricos, es decir tienen que funcionar para cualquier nº de entrada.
    - A)

```
1
22
333
4444
55555
```

    - B)

```
1
 2
  3
   4
    5
```

    - C)

```
    1
   22
  333
 4444
55555
```

    - E)

```
    1
   222
  33333
 4444444
555555555
```

20. Crea un programa que pida dos números enteros por teclado y muestre por pantalla el siguiente menú:

    MENÚ

    1. Sumar
    2. Restar
    3. Multiplicar
    4. Dividir
    5. Salir

    Elija una opción:

    El usuario elegirá el número de la opción que corresponda con la operación matemática elegida, se mostrará el resultado y se volverá a mostrar de nuevo el menú. El programa se ejecutará hasta que el usuarios elija la opción 5 (Salir). Si el usuario introduce una opción no contemplada en el menú, le mostrará un mensaje y de nuevo el menú para que seleccione una opción correcta.