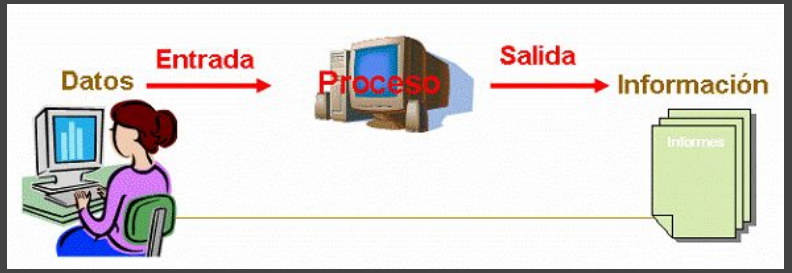
Clase 3

Repaso de lo que vimos en la **#clase2**

TIPOS DE DATOS
VARIABLES
UN POCO DE CODEO

Entrada y salida de datos



Para poder trabajar con datos es necesario saber ingresarlos a la computadora y también imprimirlos por pantalla.

Acá es necesario aclarar algo: *los programas, salvo que se rompan, se ejecutan por completo, pero por pantalla sólo vamos a ver aquello que decidimos que sea ingresado e impreso*.

Salida de datos

print()

- es una función predefinida de Python (built-in function)
- su tarea es imprimir por consola, terminal o intérprete (según lo que se esté utilizando) aquello que le pasemos por parámetro
- se puede pasar como parámetro: un dato, una variable o una función

```
print(dato)
```

Entrada de datos

input()

- es una función predefinida de Python (built-in function)
- su tarea es imprimir por consola, terminal o intérprete (según lo que se esté utilizando) aquello que se le pida al usuario que ingrese por parámetro
- dentro de los paréntesis se puede poner un string que indique al usuario qué es lo que se espera que ingrese
- todo lo que se ingresa por teclado es un string, por más que sea un número, para poder tratarlo diferente hay que convertirlo

input()

input("Ingrese un número: ")

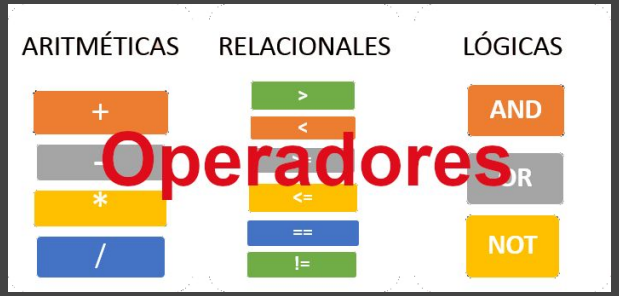
```python
a = int(input("Ingrese un número: "))
print(type(a))

b = float(input("Ingrese un número: "))
print(type(b))

c = input("Ingrese un número: ")
print(type(c))
```

```
Ingrese un número: 5
<class 'int'>
Ingrese un número: 5
<class 'float'>
Ingrese un número: 5
<class 'str'>
```

Operadores



Un operador es un *símbolo que se aplica a uno o varios operandos en una expresión o instrucción con el fin de obtener cierto resultado*.

ARITMÉTICOS | **DE ASIGNACIÓN** | **RELACIONALES** | **LÓGICOS** | **A NIVEL DE BIT**

(la maga de Python)

ARITMÉTICOS

Símbolos que se utilizan para realizar operaciones aritméticas y manipular datos de tipo int, float y complex, a excepción de + que se utiliza también para concatenar strings.

- \+ SUMA
- \- RESTA/NEGACIÓN
- * PRODUCTO
- ** POTENCIA
- / DIVISIÓN
- // DIVISIÓN ENTERA
- % MÓDULO

(la maga de Python)

DE ASIGNACIÓN

Símbolos que se utilizan para asignar un valor a una variable.

- = ASIGNA EL VALOR DE LA DCHA A LA VARIABLE DE LA IZQ
- += SUMA A LA VARIABLE DE LA IZQ EL VALOR DE LA DCHA
- -= RESTA A LA VARIABLE DE LA IZQ EL VALOR DE LA DCHA
- *= MULTIPLICA A LA VARIABLE DE LA IZQ POR EL VALOR DE LA DCHA
- /= DIVIDE A LA VARIABLE DE LA IZQ POR EL VALOR DE LA DCHA

- **= ELEVA A LA VARIABLE DE LA IZQ A LA POTENCIA DE LA DCHA
- //= DIVIDE DE MANERA ENTERA A LA VARIABLE DE LA IZQ POR LA VARIABLE DE LA DCHA
- %= DEVUELVE EL RESTO DE LA DIVISIÓN ENTRE LA VARIABLE DE LA IZQ Y LA DE LA DCHA

RELACIONALES

Aquellos operadores que, como su nombre lo indica, *devuelven un valor booleano según la relación que haya entre los operandos.*

Todos darán como resultado True si efectivamente cumplen su tarea, de lo contrario darán como resultado False.

- == COMPARA QUE DOS VALORES SEAN EXACTAMENTE IGUALES
- != COMPARA SI DOS VALORES SON DISTINTOS
- < EVALÚA SI EL VALOR DE LA IZQ ES MENOR QUE EL DE LA DCHA
- > EVALÚA SI EL VALOR DE LA IZQ ES MAYOR QUE EL DE LA DCHA
- <= EVALÚA SI EL VALOR DE LA IZQ ES MENOR O IGUAL QUE EL DE LA DCHA
- >= EVALÚA SI EL VALOR DE LA IZQ ES MAYOR O IGUAL QUE EL DE LA DCHA

LÓGICOS

Aquellos que *devuelven un resultado booleano si se cumple la función del operador. Se aplican sobre valores booleanos*.

- and EVALÚA SI TODOS LOS OPERANDOS INVOLUCRADOS SON TRUE
- or EVALÚA SI AL MENOS UNO DE LOS OPERANDOS INVOLUCRADOS ES TRUE
- not CAMBIA EL VALOR DEL OPERANDO INVOLUCRADO

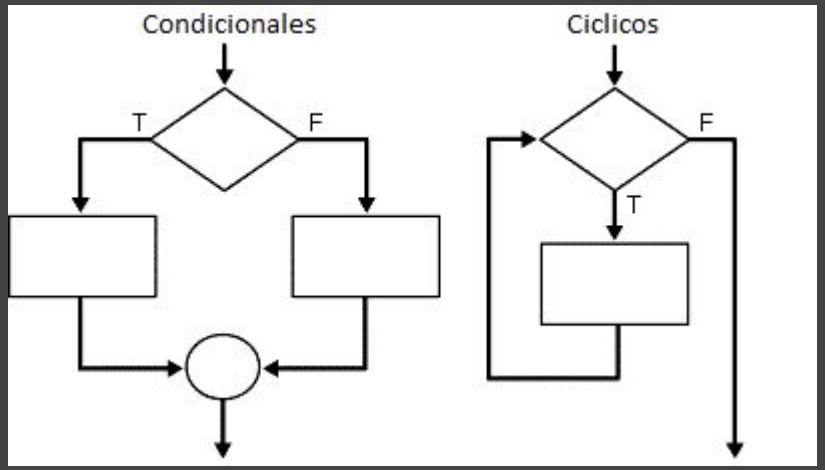
A NIVEL DE BIT

Las computadoras reciben información y la procesan en binario (unos y ceros), cada uno y cada cero que forme parte de un número o de una palabra es un bit. Por ejemplo, el número binario 1000 (8 en decimal), ocupa 4 bits y 1 byte. Un byte son ocho bits, y siempre se completa el octeto con ceros a la izquierda.

Por consiguiente, *también hay operadores para trabajar con estos datos*.

- & COMPARA LOS BITS DE CADA OPERANDO Y DEVUELVE 1 CUANDO AMBOS SON 1 Y 0 EN OTRO CASO, FORMANDO OTRO BINARIO Y CONVIRTIÉNDOLO A DECIMAL (BASE 10)
- | COMPARA LOS BITS DE CADA OPERANDO Y DEVUELVE 1 SI AL MENOS UNO DE LOS DOS ES 1 Y 0 EN OTRO CASO, FORMANDO OTRO BINARIO Y CONVIRTIÉNDOLO A DECIMAL (BASE 10)
- ~ CAMBIA EL VALOR DE CADA BIT, FORMANDO OTRO BINARIO Y CONVIRTIÉNDOLO A DECIMAL (BASE 10)
- << DESPLAZA HACIA LA IZQ LOS BITS QUE SE INDIQUEN, FORMANDO OTRO BINARIO Y CONVIRTIÉNDOLO EN DECIMAL
- >> DESPLAZA HACIA LA DCHA LOS BITS QUE SE INDIQUEN, FORMANDO OTRO BINARIO Y CONVIRTIÉNDOLO EN DECIMAL

UNIDAD V
ESTRUCTURAS DE CONTROL DE FLUJO



La mayoría de las veces que creamos y codeamos un programa, necesitamos *chequear ciertas condiciones y ver qué camino seguir y/o cuántas veces. Para esto, se utilizan las estructuras de control de flujo*.

CONDICIONALES

ITERATIVAS

(la maga de Python)

CONDICIONALES

Permiten comprobar condiciones y hacer que el programa se comporte de una forma u otra, que ejecute un fragmento de código u otro, dependiendo del valor de verdad de esa condición.

Sentencia *if*

Es la *forma más simple de crear una estructura de flujo condicional*, if, en español “si”, seguido de la o las condiciones a evaluar, y luego dos puntos. En caso de que la o las condiciones resulte/n falsa/s, el programa seguirá a la próxima instrucción, dejando sin efecto el código dentro del if.

```
if condición:

    bloque de código...
```

Sentencia *if-else*

Muchas veces, si la condición es falsa, no vamos a querer que el programa continúe a la siguiente instrucción sino que ejecute un código particular. Para eso existe la sentencia if-else, en español si-si no. *Si la condición es verdadera se ejecuta el código luego del if, si es falsa, se ejecuta el código luego del else*.

```
if condición:
    bloque de código...
else:
    bloque de código...
```

Sentencia *if-elif*

En otros lenguajes elif sería else if, es decir, *es para crear un else (si la condición es falsa) y agregarle una condición a ese else*. Luego puede ir un else al final o no.

```
if condición:
    bloque de código...
elif condición2:
    bloque de código...
else:
    bloque de código...
```

Sentencias if anidadas: *if-if-if...*

Estas sentencias *se utilizan cuando, luego de comprobar si una condición es True, se desea comprobar otra u otras condicione/s.* Es decir, cada subcondición depende de la verdad de la anterior para ser comprobada o no.


```
if condición:
    if condición2:
        bloque de código…
        if condición3:
            bloque de código...
        else:
            bloque de código...
```

ITERATIVAS

Las estructuras de control de flujo iterativas permiten ejecutar un mismo fragmento de código un cierto número de veces, mientras se cumpla una determinada condición. Estas estructuras son a menudo llamadas bucles.

Bucle *while*

El bucle while, en español mientras, *ejecuta un fragmento de código mientras cierta condición permanece verdadera*.

```
while condición:

    bloque de código...
```

Bucle *for*

A diferencia de otros lenguajes de programación, en Python *for se utiliza para recorrer objetos iterables*.

Un objeto iterable es aquel que puede ser recorrido elemento a elemento, por ejemplo una lista, una tupla, un diccionario.

```
for elemento in objeto:

    bloque de código...
```

Sentencias *pass, break & continue*

- pass: *devuelve NULL al leer el interior de un bucle,* como si estuviera vacío (que puede estarlo). Se suele utilizar cuando se está creando una función pero aún no se determinó su tarea, o una clase y aún no tiene atributos y métodos propios.
- break: *termina el bucle actual y salta a la siguiente instrucción del programa*.
- continue: *salta a la siguiente iteración del bucle, ignorando la actual*.