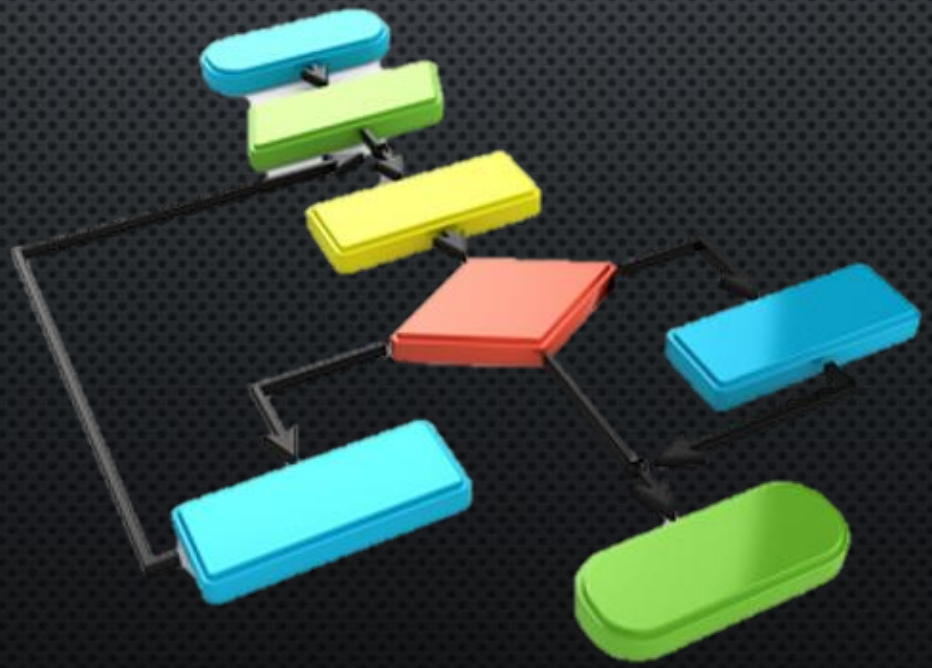
PYTHON

CONTROL DE FLUJO

En los lenguajes como Python, el ordenador sigue la secuencia de instrucciones del código y lo ejecuta línea a línea.

Denominamos flujo de un programa al orden en la ejecución del código, este orden secuencial puede alterarse

En esta clase examinaremos qué secuencias o instrucciones ofrece Python para esto.



SENTENCIA CONDICIONAL IF

Es la opción más conocida para controlar el flujo de un programa. Las condiciones permiten elegir entre caminos distintos según el valor de una expresión.

Pseudocódigo

```
Si edad >= 18 entonces
   Mostrar “Entrar al boliche”
Fin_si
```

En Python

```
if  edad >= 18:
   print(“Entrar al boliche”)
```

El bloque de código asociado a la condición se inicia después de : con un sangrado (indentado) que determina el bloque de código.

Todas las instrucciones que pertenecen a un mismo bloque deben tener idéntico sangrado.

El bloque termina cuando el sangrado vuelve a la posición de inicio if.

Recuerden que Python usa el sangrado para identificar los bloques de código.

SENTENCIA CONDICIONAL IF DOBLE

Para construir esta estructura en Python, la sentencia if puede hacer uso de la palabra reservada else, la cual permite definir otro bloque de código alternativo.

```
Pseudocódigo

Si edad >= 18 entonces
   Mostrar “Mayor de edad”
Sino
   Mostrar “Menor de edad”
Fin_si
```

```
Python

if edad >= 18:
   print(‘Mayor de edad’)
else:
   print(‘Menor de edad’)
```

SENTENCIA CONDICIONAL IF ANIDADA

Otra posibilidad es realizar varias comprobaciones seguidas, donde el programa debe una acción sobre la base de valores determinados de una variable.

Imaginemos que estamos construyendo un programa para un robot clasificador de huevos por tamaño. Nuestro brazo robot recibe la información de una balanza, la cual le indica en gramos , el peso del huevo que debe clasificar. El brazo debe, a partir del peso, ubicar el huevo en una u otra caja como sigue:
Caja “S”: peso menor a 53 gramos.
Caja “M”: peso mayor o igual a 53 gramos e inferior a 63 gramos
Caja “L”: peso mayor o igual a 63 gramos e inferior a 73 gramos
Caja “XL”: peso mayor o igual a 73 gramos.

Con lo que hemos visto hasta aquí, podemos resolver el problema, gracias a la posibilidad de anidar estructuras.

```
if peso < 53:
   caja = ‘S’
else:
   if peso < 63:
      caja = ‘M’
   else:
      if peso < 73:
         caja = ‘L’
      else:
         caja = ‘XL’
```

Existe una solución mejor cuando encadenamos sentencias else-if , Python ofrece la sentencia elif para reemplazarlas

```
if peso < 53:
   caja = ‘S’
elif peso < 63:
      caja = ‘M’
   elif peso < 73:
         caja = ‘L’
      else:
         caja = ‘XL’
```

Esta es también la alternativa que ofrece Python a estructuras de tipo Switch-Case que utilizan otros lenguajes

SENTENCIA REPETITIVA WHILE

Cuando nos encontramos ante la necesidad de repetir determinadas instrucciones mientras se cumpla una condición, la solución es siempre la sentencia while.

```
print('Ingrese el peso del huevo ')
peso = float(input())
while peso != 0:
    if peso < 53:
        caja = 'S'
    elif peso < 63:
        caja = 'M'
    elif peso < 73:
        caja = 'L'
    else:
            caja = 'XL'
    print('Deberá colocar el huevo en la caja: ',caja)
    print('Ingrese el peso del huevo ')
    peso = float(input())
```

Ahora veremos cómo se calcula el factorial de un número natural. El mismo resulta demultiplicar ese número por todos los números inferiores a él hasta el 1.
Así 4! = 4 * 3 * 2 * 1

```
print('Ingrese un número para calcular el factorial ')
factorial = int(input())
acumula = 1
while factorial > 1:
   acumula = acumula * factorial
   factorial = factorial - 1
print('El factorial es = ',acumula)
```

SENTENCIA REPETITIVA FOR

En Python la sentencia for puede tomar dos formas:

a) Secuencia definida mediante una lista

Se puede especificar una secuencia mediante una lista de valores escritos entre corchetes [ ] o entre paréntesis ( ) con el siguiente formato:

```
for  v in secuencia:
        instrucción en el bloque p
        instrucción en el bloque p
        . . .
        instrucción en el bloque p
```

En donde v es la variable que recorre la secuencia y p es el bloque que contiene las instrucciones que se desea repetir

Las siguientes listas son ejemplos de secuencias de control para el ciclo for:

[2, 5, 7, 8, 9, 5, 4]

[‘Enero’, ‘Febrero’, ‘Marzo’, ‘Abril’, ‘Mayo’]

[3.5, 2.8, 0.75, 4.3, 7.6]

Describir un ciclo para mostrar cada elemento entre 1 y 5 junto con su cuadrado

```
for n in [1,2,3,4,5]:
    c = n ** 2
    print(n,'^ 2 = ',c)
```

Describir un ciclo que muestre los nombres de los departamentos de la provincia de Mendoza.

```
for d in ['Las Heras', 'Lavalle', 'Luján', 'Tupungato', 'Tunuyán',
          'San Rafael', 'General Alvear', 'Malargüe']:

    print('Departamento: ', d)
```

b) Secuencia definida mediante un rango

La forma más utilizada para especificar secuencias de control para la estructura for se define con la función range, la cual permite establecer un rango de valores.

```
for v in range(especificación):
    instrucción en el bloque p
    instrucción en el bloque p
    . . .
    instrucción en el bloque p
```

En donde v es la variable que tomará cada valor del rango especificado y p es el bloque que contiene las instrucciones que se desean repetir.
El rango se debe especificar únicamente con números enteros, no se pueden usar decimales o caracteres.

El rango se puede especificar con el valor final, En este caso el valor inicial es cero.

range(10) → genera los valores 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 el ciclo se repetirá 10 veces

El rango se puede especificar con los extremos

range(2, 10) → genera los valores 2, 3, 4, 5, 6, 7, 8, 9 el ciclo se repetirá 8 veces.

range(4, 1) → el valor inicial es mayor que el valor final por lo que no se pueden generar valores. El ciclo no se repite ni una sola vez

La secuencia generada con range no incluye el extremo final del rango

El rango se puede especificar con los extremos y el incremento

range(1, 15, 2) → genera los valores 1, 3, 5, 7, 9, 11, 13 el ciclo se repite 6 veces

range(8, 1, -1) → genera valores 8, 7, 6, 5, 4, 3, 2

Se necesita sumar los cuadrados de los primeros n números naturales

```
# suma de cuadrados
n = int(input('Ingrese el valor de final '))
s = 0
for i in range(1,n+1):
    c=i**2
    s=s+c
print('La suma es:', s)
```

Calcule y muestre el promedio de un grupo de datos ingresados desde el teclado:

```
# promedio de un grupo de datos
n = int(input('Indique la cantidad de datos a ingresar: '))
suma = 0
for i in range(n):
    x=float(input('Ingrese el siguiente valor: '))
    suma = suma + x
promedio=suma/n
print('El promedio de los valores ingresados es:
',promedio)
```

CICLOS ANIDADOS

Como ocurre con cualquier estructura, los bucles también pueden anidarse. Para cada iteración del bucle externo, se producen todos los ciclos del bucle interno.

Liste todas las parejas de números del 1 al 3

```
# parejas de números

for i in range(1, 4):
    for j in range(1, 4):
        print(i,j)
```

CONTROL CON BREAK Y CONTINUE

La sentencia break permite romper un bucle en cualquier momento, terminar las iteraciones y continuar después del mismo. Generalmente usaremos break después de realizar alguna comprobación que motive abandonar una repetición while o for.

Simular lanzamientos de un dado. Determinar la cantidad de intentos realizados hasta que salga un 5.

```
# uso de break
from random import*
n = int(input('cantidad máxima de lanzamientos: '))
for i in range(n):
    dado = randint(1, 6)
    print(dado)
    if dado == 5:
        print('Salio el 5!! en el lanzamiento',i+1)
        break
```

La instrucción break rompe el flujo dentro del cuerpo de instrucciones del bucle para abandonarlo. En cambio, la instrucción continue rompe el flujo para saltar a la siguiente iteración.

Supongamos un mes de 31 días, con el 1 en lunes. El siguiente programa indica qué hacer cada día teniendo en cuenta que los sábados y domingos no son laborables.

```
# uso de continue.
for dia in range(1, 31):
    print('---------')
    print('Día ',dia)
    if dia % 7== 6 or dia % 7 == 0:
        print('DESCANSAR')
        continue
    print('LEVANTARSE TEMPRANO')
    print('HAY QUE ESTUDIAR')
```

El módulo random es una librería de Python que contiene funciones que permiten generar números aleatorios. Para acceder a este módulo se lo debe cargar con la siguiente directiva:

```
from  random import*
```

Algunas funciones básicas del módulo random:

- Generar un número aleatorio real en el intervalo semi abierto [0, 1)
  `random( )`
- Generar un número aleatorio entero en el intervalo [a,b] invluyendo los extremos
  `Randint(a,b)`
- Iniciar una secuencia de números aleatorios con una semilla dada
  `seed(n)`

La instrucción pass

La sentencia pass no hace nada, aunque resulta útil en varias situaciones. En Python no podemos tener sin definir un bloque de código, por ejemplo, el cuerpo de una función, el cuerpo de una condición o de un bucle. Es habitual usar pass cuando estamos escribiendo la estructura de nuestro programa pero aún no hemos abordado la implementación de determinados bloques de código.

```
if contador == 0:
    pass # por hacer
```

Generar un menú para el usuario

Los programas en los cuales el usuario interactúa con la computadora de manera continua se denominan interactivos. Esta interacción es múltiple en muchas aplicaciones. En su forma más elemental se usa un menú para que el usuario elija una opción. Para cada opción, la computadora realiza una acción.

La siguiente fórmula permite convertir un valor de temperatura entre grados Farenheit (f) a Celcius (c) $c = \frac{5}{9}(f - 32)$

Escribir un programa con un menú para la conversión en ambos casos.

```
# uso menues. Conversión de temperaturas
salida = False
while not salida:
    print('1: convertir F a C')
    print('2: Convertir C a F')
    print('3: Salir')
    opcion = input('Elija opcion')
    if opcion == '1':
        f = int(input('Ingrese grados F'))
        c = 5 / 9 * (f - 32)
        print(c)
    elif opcion == '2':
        c = int(input('Ingrese grados C'))
        f = 9 / 5 * c + 32
        print(f)
    elif opcion == '3':
        salida = True
```