

PYTHON



Innovación
en Formación
Profesional



UT01 - INTRODUCCIÓN

- 1. CARACTERÍSTICAS DE PYTHON
- 2. EMPEZANDO A TRABAJAR CON PYTHON
 - 2.1. Sintaxis Básica
 - 2.2. Tipos de datos
- 3. FUNCIONES
- 4. LISTAS
- 5. TUPLAS
- 6. CONDICIONALES
 - 6.1. IF
 - 6.2. IF – ELSE IF
 - 6.3. ELIF
 - 6.4. COMPARADORES CONCATENADOS
- 7. TUPLAS
- 8. BUCLES
- 8. GENERADORES

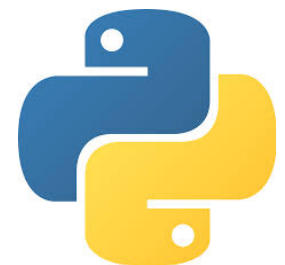
■ 1. INTRODUCCIÓN A PHYTON

UT 1

Introducción a Python

1. ¿QUÉ ES PYTHON?

- Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en la legibilidad de su código.
- Es un lenguaje de programación multiparadigma, ya que soporta programación orientada a objetos, programación imperativa (ensamblador) y programación funcional.
- Creado por Guido Van Rossum a comienzos de los 90.

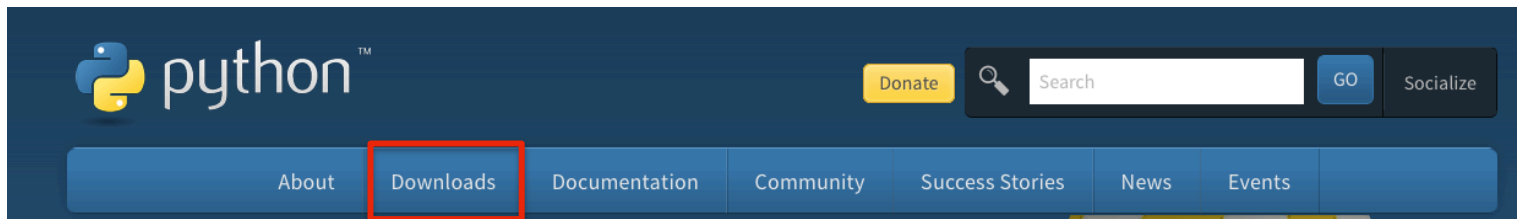


1. ¿POR QUÉ PYTHON?

- Amplia comunidad de desarrolladores.
- Open Source
- Multiparadigma (programación imperativa / programación funcional)
- Multidisciplinar (desarrollo web, big data, inteligencia artificial, videojuegos...)
- Multiplataforma
- Lenguaje de alto nivel.
- Tipado dinámico y fuerte.
- Orientado a objetos.
- Fácil de aprender.
- Librería estándar muy amplia.
- Interpretado.

2. SOFTWARE

- PYTHON → <https://www.python.org>

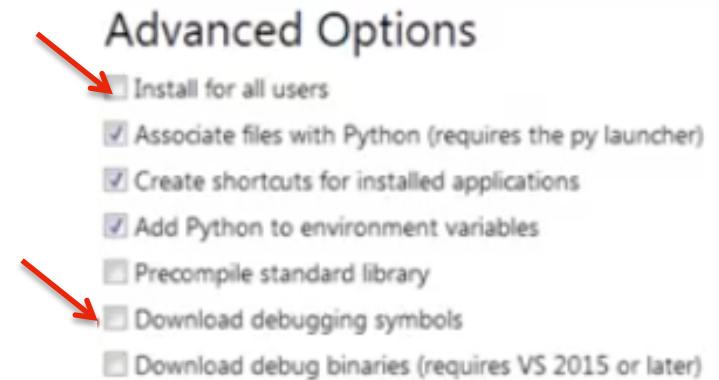
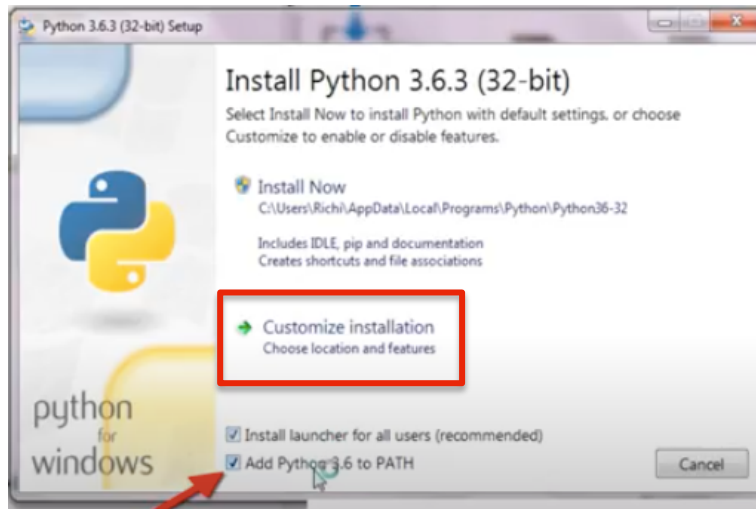


- PYCHARM (Versión Community)
 - Podemos descargarlo directamente de <https://www.jetbrains.com/es-es/pycharm/>



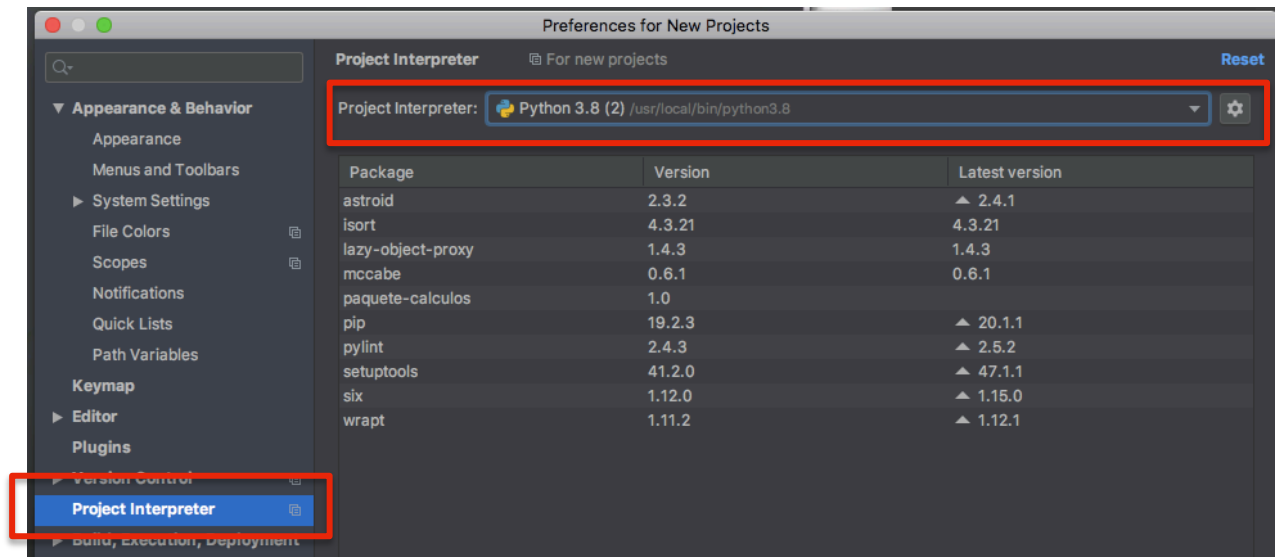
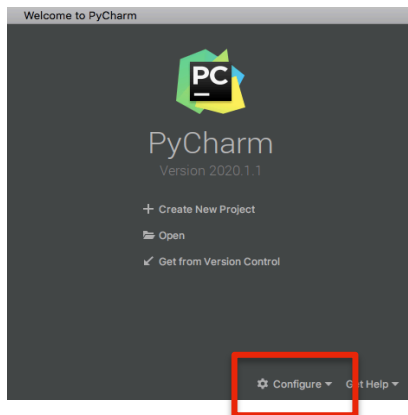
2.1. Instalación de Python

- Una vez descargado procedemos a su instalación:



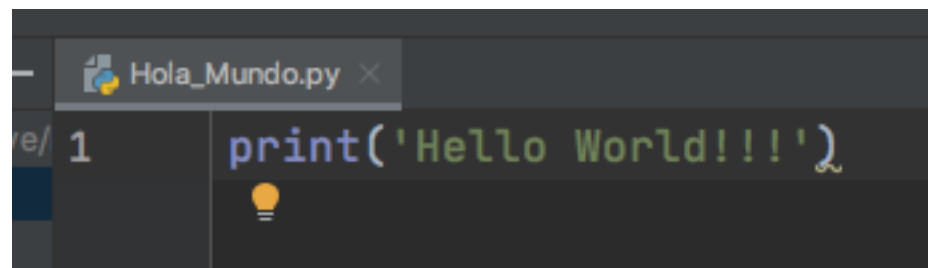
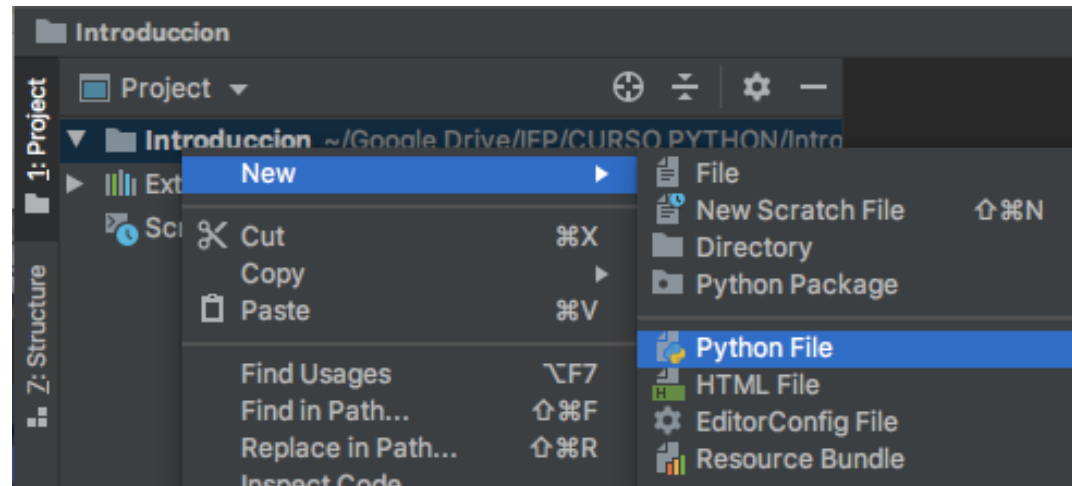
2.1. Configuración de Pycharm

Una vez instalado Pycharm, debemos configurar el intérprete de Python, indicándole que use el que acabamos de descargar.



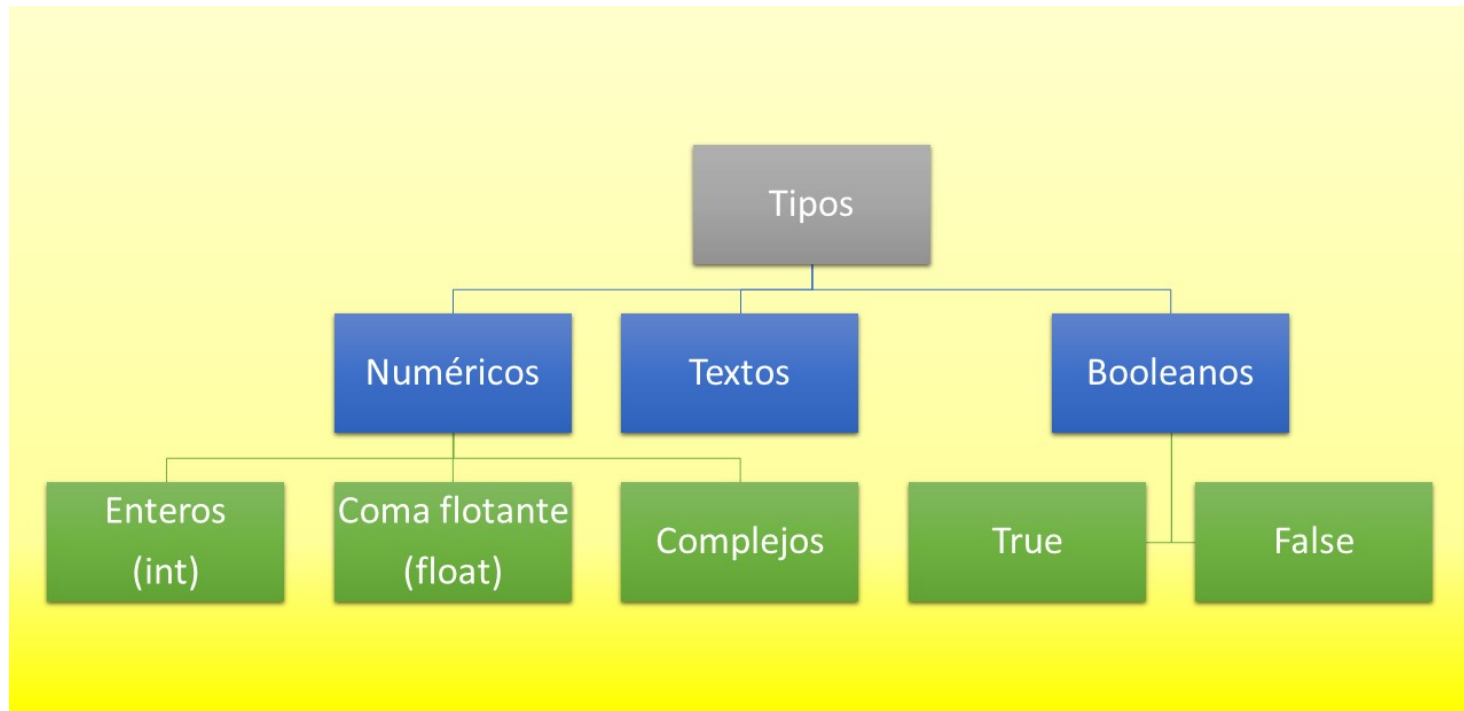
3. CREACIÓN DE UN NUEVO PROYECTO

- Creamos un nuevo proyecto en Pycharm y le agregamos un nuevo archivo Python.



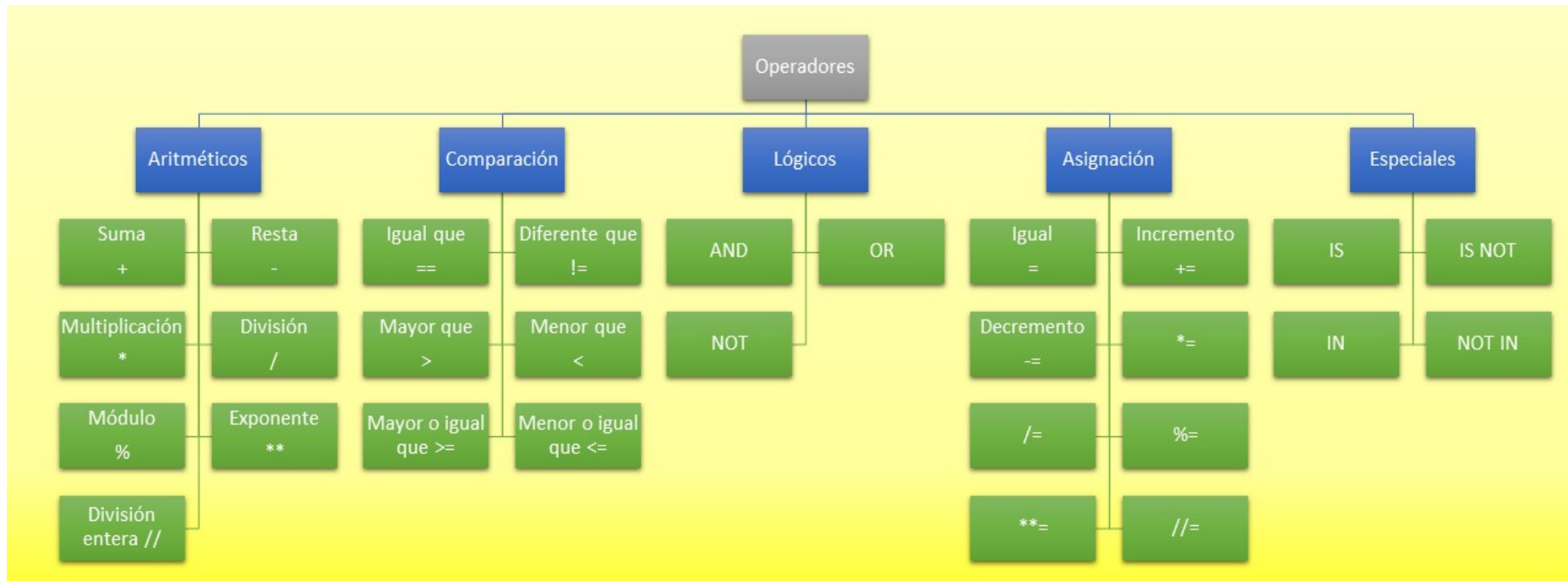
4. SINTAXIS BÁSICA

En Python vamos a manipular básicamente tres tipos de datos:



4. SINTAXIS BÁSICA

En Python los operadores se dividen en cinco categorías:



4. SINTAXIS BÁSICA

```
1  #escritura por consola
2  print("Hola Mundo")
3  print("Adiós")
4
5  #variables
6  nombre="Silvia"
7  print(nombre)
8
9  #tipos de variable
10 print(type(nombre))
```

Comentarios #

En Python no se declara el tipo de variable, es el contenido el que establece qué tipo de variable es.

Python es un lenguaje 100% orientado a objetos, incluso las variables son objetos.

4. SINTAXIS BÁSICA

A diferencia de otros lenguajes Python permite usar comillas triples para escribir cadenas de texto que ocupen varias líneas.

```
-#variables de tipo texto con varios
-#saltos de línea

mensaje="""Esto es un mensaje
con tres saltos
de línea"""
print(mensaje)
```

5. FUNCIONES

¿QUÉ SON?

- Conjunto de líneas de código agrupadas que funcionan como una unidad realizando una tarea específica.
- Las funciones en Python pueden devolver valores.
- Las funciones en Python pueden tener parámetros/argumentos.
- A las funciones también se les llama “métodos” cuando se encuentran dentro de una clase.

5. FUNCIONES

¿UTILIDAD?

- Reutilización de código

SINTAXIS

- **def** nombre_función()
 - Instrucciones de la función
 - return (opcional)
 - def nombre_función(parámetros)
 - Instrucciones de la función
 - return (opcional)
- PALABRA RESERVADA
- ZONA DE PARÁMETROS

5. FUNCIONES

EJECUCIÓN

- Nombre_funcion()
- Nombre_funcion(parámetros)
- Una función no hará nada hasta que no sea llamada.

DECLARACIÓN

```
def mensaje():
```

```
    print("Estamos aprendiendo Python")  
    print("funciones basicas")  
    print("es facil")
```

CUERPO.
IDENTADO!!!

```
mensaje()
```

LLAMADA A
FUNCIÓN

5. FUNCIONES

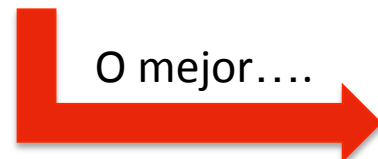
FUNCIONES CON PARÁMETROS

Vamos a crear una función que sume dos números.

```
def suma():  
    num1=5  
    num2=7  
    print(num1+num2)  
  
suma()
```



```
def suma(num1, num2):  
    print(num1+num2)  
  
suma(5,7)  
suma(10,4)  
suma(6,9)
```



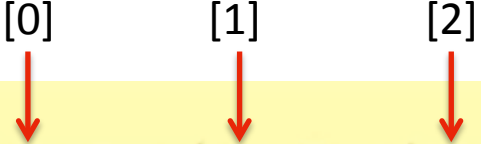
O mejor....

```
def suma(num1, num2):  
    resultado=num1+num2  
    return resultado  
  
print(suma(5,65))  
print(suma(10,4))  
print(suma(6,9))
```

6. LISTAS

- Estructuras de datos que permiten almacenar gran cantidad de valores (arrays en otros lenguajes)
- En Python las listas pueden guardar diferentes tipos de valores.
- Se pueden expandir dinámicamente añadiendo nuevos elementos.

SINTAXIS



```
nombreLista=[elem1, elem2, elem3.....]
```

6. LISTAS

Trabajando con listas...

```
mi_Lista = ["Silvia", "Pepe", "Marta", "Antonio"]  
print(mi_Lista[:])
```

Muestra todos los
elementos de la
lista

```
mi_Lista = ["Silvia", "Pepe", "Marta", "Antonio"]  
print(mi_Lista[2])
```

Accede a un
elemento en
concreto

```
mi_Lista = ["Silvia", "Pepe", "Marta", "Antonio"]  
print(mi_Lista[-2])
```

Con números negativos,
empezamos a contar desde el final
del array, desde la posición 1

6. LISTAS

Trabajando con listas...

```
mi_Lista = ["Silvia", "Pepe", "Marta", "Antonio"]  
print(mi_Lista[0:3])
```

Muestra todos los tres primeros elementos de la lista, incluye la posición 0, pero no la 3

```
mi_Lista = ["Silvia", "Pepe", "Marta", "Antonio"]  
print(mi_Lista[:3])
```

Muestra todos los tres primeros elementos de la lista, incluye la posición 0, pero no la 3

```
mi_Lista = ["Silvia", "Pepe", "Marta", "Antonio"]  
print(mi_Lista[2:])
```

Accede a los elementos que hay desde el que está en la posición 2 hasta el final

6. LISTAS

Agregar elementos a listas...

```
mi_Lista = ["Silvia", "Pepe", "Marta", "Antonio"]  
mi_Lista.append("Sandra")  
print(mi_Lista[:])
```

Agrega un elemento al final de la lista

```
mi_Lista = ["Silvia", "Pepe", "Marta", "Antonio"]  
mi_Lista.insert(2, "Sandra")  
print(mi_Lista[:])
```

Inserta un elemento en la posición indicada en el primer argumento

```
mi_Lista = ["Silvia", "Pepe", "Marta", "Antonio"]  
mi_Lista.extend(["Sandra", "Carlos", "Ana", "Lucía"])  
print(mi_Lista[:])
```

Para concatenar una lista de elementos al final

6. LISTAS

Agregar elementos a listas...

```
mi_Lista = ["Silvia", "Pepe", "Marta", "Antonio"]
mi_Lista.extend(["Sandra", "Carlos", "Ana", "Lucia"])
print(mi_Lista.index("Antonio"))
```

Devuelve el índice de la posición en la que se encuentra el argumento. Si un elemento está repetido, devuelve la posición del primer elemento

```
mi_Lista = ["Silvia", "Pepe", "Marta", "Antonio"]
mi_Lista.extend(["Sandra", "Carlos", "Ana", "Lucia"])
print("Pepe" in mi_Lista)
```

Devuelve True si el argumento está en la lista, False si no lo está

```
mi_Lista = ["Silvia", 5, 25.43, True]
mi_Lista.extend(["Sandra", False, "Ana", "Lucia"])
print(mi_Lista[1])
```

En Python una Lista puede contener valores de distintos tipos

6. LISTAS

Eliminar elementos de listas...

```
mi_Lista = ["Silvia", 5, 25.43, True]
mi_Lista.extend(["Sandra", False, "Ana", "Lucia"])
mi_Lista.remove("Ana")
print(mi_Lista[:])
```

Elimina el elemento indicado de la lista

```
mi_Lista = ["Silvia", 5, 25.43, True]
mi_Lista.extend(["Sandra", False, "Ana", "Lucia"])
mi_Lista.pop()
print(mi_Lista[:])
```

Elimina el último elemento de la Lista

6. LISTAS

Concatenar listas...

```
mi_Lista = ["Silvia", 5, 25.43, True]
mi_Lista2 = ["Sandra", "Lucia"]
mi_Lista3 = mi_Lista+mi_Lista2
print(mi_Lista3)
```

El signo + concatena listas

```
mi_Lista = ["Silvia", 5, 25.43, True] * 3
print(mi_Lista[:])
```

Imprime la lista 3 veces

```
lista1=[3,4,1,2]
lista1.sort()
print(lista1)
```

Ordena la lista



7. TUPLAS

- Las tuplas son listas inmutables, es decir, no se pueden modificar después de su creación.
- No permiten añadir, eliminar, mover elementos... (no append, extend, remove...)
- Permiten extraer porciones, pero el resultado es una tupla nueva
- Sí permiten comprobar si un elemento está en la tupla

¿VENTAJAS?

- Más rápidas
- Menos espacio
- Formatean Strings
- Pueden utilizarse como claves en un **diccionario**

7. TUPLAS

SINTAXIS

```
nombreLista=(elem1, elem2, elem3.....)
```

```
mitupla = ("Juan", 22, 1, 1984)  
print(mitupla[2])
```

Acceder a un elemento
concreto de una tupla

7. TUPLAS

CONVERTIR UNA TUPLA EN UNA LISTA

```
mitupla = ("Juan", 22, 1, 1984)
milista = list(mitupla)

print(milista[:])
```

CONVERTIR UNA LISTA EN UNA TUPLA

```
milista = ["Juan", 22, 1, 1984]
mitupla = tuple(milista)

print(mitupla)
```

7. TUPLAS

```
milista = ["Juan", 22, 1, 1984]  
mitupla = tuple(milista)  
  
print("Juan" in mitupla)
```

Devuelve true si el elemento se encuentra en la tupla

```
milista = ["Juan", 22, 1, 1984]  
mitupla = tuple(milista)  
  
print(mitupla.count(22))
```

Devuelve el número de veces que se repite un elemento en una tupla

```
milista = ["Juan", 22, 1, 1984]  
mitupla = tuple(milista)  
  
print(len(mitupla))
```

Devuelve el número de elementos que hay dentro de la tupla

7. TUPLAS

```
mitupla = ("Silvia",)  
print(len(mitupla))
```

Crea una tupla unitaria. OJO!! El elemento que la compone debe ir seguido de una coma!!

DESEMPAQUETADO DE TUPLA

```
mitupla = ("Juan", 8, 1977, "junio")  
nombre, dia, anyo, mes=mitupla  
  
print(nombre)  
print(dia)  
print(mes)  
print(anyo)
```

Guarda dentro de variables cada uno de los elementos de la tupla.

7. TUPLAS

ACTIVIDADES

1. Mete los valores del 1 al 100 en una lista.

```
lista = []  
  
i=1  
while i<=100:  
    lista.append(i)  
    i=i+1  
  
print (lista)
```



8. DICCIONARIOS

- **¿QUÉ SON?**
- Estructuras de datos que nos permiten almacenar valores de diferentes tipos e incluso listas y otros diccionarios.
- Su principal característica es que los datos se almacenan asociados a una clave de tal forma que se crea una asociación de tipo clave : valor para cada elemento almacenado.
- Los elementos almacenados no están ordenados. El orden es indiferente a la hora de almacenar información en un diccionario.

8. DICCIONARIOS

- **CASO PRÁCTICO**
- Queremos crear un diccionario que almacene países y capitales.

Clave Valor

```
midiccionario={"Alemania":"Berlin", "Espana":"Madrid", "Francia":"Paris", "Reino Unido":"Londres"}  
print(midiccionario["Francia"])
```

Si preguntamos por la clave, nos devolverá su valor correspondiente

8. DICCIONARIOS

■ AGREGAR ELEMENTOS AL DICCIONARIO

- Para agregar un elemento a un diccionario, procederemos de la siguiente forma

```
midiccionario={"Alemania":"Berlin","Espana":"Madrid", "Francia":"Paris", "Reino Unido":"Londres"}  
midiccionario["Italia"]="Lisboa"  
print(midiccionario)
```

■ SOBREScribir UN VALOR

```
midiccionario={"Alemania":"Berlin","Espana":"Madrid", "Francia":"Paris", "Reino Unido":"Londres"}  
midiccionario["Italia"]="Lisboa"  
print(midiccionario)  
midiccionario["Italia"]="Roma"  
print(midiccionario)
```

8. DICCIONARIOS

■ ELIMINAR UN ELEMENTO DE UN DICCIONARIO

```
midiccionario={"Alemania":"Berlin","Espana":"Madrid", "Francia":"Paris", "Reino Unido":"Londres"}
midiccionario["Italia"]="Lisboa"
print(midiccionario)
midiccionario["Italia"]="Roma"
print(midiccionario)
del midiccionario["Reino Unido"]
print(midiccionario)
```

8. DICIONARIOS

- **CREAR DICIONARIOS CON DIFERENTES TIPOS DE DATOS**

```
midiccionario={"España":"Madrid", 23:"Jordan", "Meses":12}
```

- **CREAR UN DICIONARIO MEDIANTE UNA LISTA**

```
mitupla=["Espana", "Francia", "Reino Unido", "Alemania"]  
midiccionario={mitupla[0]:"Madrid", mitupla[1]:"Paris", mitupla[2]:"Londres", mitupla[3]:"Berlin"}  
print(midiccionario["Francia"])
```

8. DICCIONARIOS

- ¿Y si queremos que un diccionario almacene directamente una tupla?

```
midiccionario={23:"Jordan", "Nombre":"Michael", "Equipo":"Chicago", "anillos":[1991,1992,1993,1996,1997,1998]}  
print(midiccionario["anillos"])
```

- Guardar un diccionario dentro de otro diccionario

```
midiccionario={23:"Jordan", "Nombre":"Michael", "Equipo":"Chicago",  
"anillos":{"temporadas":[1991,1992,1993,1996,1997,1998]}}  
print(midiccionario["anillos"])
```

8. DICCIONARIOS

■ MÉTODOS DE UTILIDAD DE LOS DICCIONARIOS

```
midiccionario={23:"Jordan", "Nombre":"Michael", "Equipo":"Chicago",  
"anillos":{"temporadas":[1991,1992,1993,1996,1997,1998]}}  
  
print(midiccionario.keys())  
print(midiccionario.values())  
print(len(midiccionario))
```

Imprime las claves

Imprime los valores

Devuelve el número de elementos del diccionario

8. DICCIONARIOS

■ EJERCICIO PROPUESTO

Crea un diccionario donde la clave sea el nombre del usuario y el valor sea el teléfono. Tendrás que ir pidiendo contactos hasta el usuario diga que no quiere insertar mas. No se podrán meter nombres repetidos.

```
agenda = {}  
salir = False  
  
while (not salir):  
    #Pedimos los datos  
    nombre=input("Introduce un nombre: ")  
    telefono=int(input("Introduce un telefono: "))  
  
    #Comprobamos si esta dentro del diccionario  
    if(nombre not in agenda):  
        #Añadimos el contacto  
        agenda[nombre] = telefono  
        print('Añadido el contacto')  
    else:  
        print('El nombre esta repetido')  
  
    #Indicamos si queremos salir  
    respuesta = input("¿Quieres salir? (S/N)")  
  
    if(respuesta == "S"):  
        salir = True  
  
#Mostramos el diccionario  
print(agenda)
```

9. CONDICIONALES

- IF
- IF ... ELSE
- IF... ELIF ... ELSE
- SWITCH???
- CONCATENACIÓN DE OPERADORES DE COMPARACIÓN
- OPERADORES LÓGICOS “AND” Y “OR”
- OPERADOR “IN”

9. CONDICIONALES

CONDICIONAL IF

```
print("Programa de evaluacion de alumnos")

nota_alumno=input("Introduce la nota del alumno:")

def evaluacion(nota):
    valoracion="aprobado"
    if nota<5:
        valoracion="suspense"
    return valoracion

print(evaluacion(int(nota_alumno)))
```

Hacemos un casting para convertir un texto a entero

9. CONDICIONALES

CONDICIONAL IF y ELSE

```
print("Verificacion de acceso")

edad_usuario=int(input("Introduce tu edad:"))

if edad_usuario<18:
    print("No puedes pasar")
else:
    print("Puedes pasar")

print("El programa ha finalizado")
```

CONDICIONAL ELIF

```
if edad_usuario<18:
    print("No puedes pasar")
elif edad_usuario>100:
    print("Edad incorrecta")
else:
    print("Puedes pasar")

print("El programa ha finalizado")
```

9. CONDICIONALES

CONDICIONAL IF, ELIF Y ELSE

Realiza un programa que evalúe la nota de los alumnos:

- Si la nota es menor que 5 → insuficiente.
- Si la nota es menor que 6 → suficiente
- Si la nota es menor que 7 → bien
- Si la nota es menor que 9 → notable
- Si la nota es superior o igual a 9 → sobresaliente

```
print("Verificacion de notas")

nota_usuario=int(input("Introduce tu nota:"))

if nota_usuario<5:
    print("Insuficiente")
elif nota_usuario<6:
    print("Suficiente")
elif nota_usuario<7:
    print("Bien")
elif nota_usuario<9:
    print("Notable")
else:
    print("Sobresaliente")
```

10. ENTRADA DE DATOS. LA FUNCIÓN INPUT()

La función `input()` permite obtener texto escrito por teclado. Al llegar a la función, el programa se detiene esperando que se escriba algo y se pulse la tecla Intro.

```
print("¿Cómo se llama?")
nombre = input()
print(f"Me alegro de conocerle, {nombre}")
```

```
¿Cómo se llama?
Pepe
Me alegro de conocerle, Pepe
```

Si se prefiere que el usuario escriba su respuesta a continuación de la pregunta, se podría utilizar el argumento opcional `end` en la función `print()`, que indica el carácter o caracteres a utilizar en vez del salto de línea.

```
print("¿Cómo se llama? ", end="")
nombre = input()
print(f"Me alegro de conocerle, {nombre}")
```

```
¿Cómo se llama? Pepe
Me alegro de conocerle, Pepe
```

10. ENTRADA DE DATOS. LA FUNCIÓN INPUT()

De forma predeterminada, la función `input()` convierte la entrada en una cadena, aunque escribamos un número.

Si se quiere que Python interprete la entrada como un número entero, se debe utilizar la función `int()` de la siguiente manera:

```
cantidad = int(input("Dígame una cantidad en pesetas: "))  
print(f"{cantidad} pesetas son {round(cantidad / 166.386, 2)}  
euros")
```

```
Dígame una cantidad en pesetas: 500  
500 pesetas son 3.01 euros
```

ACTIVIDADES

Ejercicio 1:

- Crea un programa que pida dos números por teclado. El programa tendrá una función llamada “DevuelveMax” encargada de devolver el número más alto de los dos introducidos.

Ejercicio 2:

- Crea un programa que pida por teclado “Nombre”, “Dirección” y “Tfno”. Esos tres datos deberán ser almacenados en una lista y mostrar en consola el mensaje: “Los datos personales son: nombre apellido teléfono” (Se mostrarán los datos introducidos por teclado).

Ejercicio 3:

- Crea un programa que pida tres números por teclado. El programa imprime en consola la media aritmética de los números introducidos.

3. CONDICIONALES

SOLUCIÓN EJERCICIO 1

```
num1=(int(input("Introduce el primer número: ")))  
num2=(int(input("Introduce el segundo número: ")))  
  
def DevuelveMax (n1, n2):  
    if n1 < n2:  
        print (n2)  
    elif n2 < n1:  
        print (n1)  
    else:  
        print ("Son iguales")  
  
print("El número más alto es: ")  
DevuelveMax(num1, num2)
```

6. CONDICIONALES

SOLUCIÓN EJERCICIO 2

```
Nombre=input("Introduce el nombre: ")
Direccion=input("Introduce la dirección: ")
Tfno=input("Introduce el teléfono: ")
listaDatos=[Nombre, Direccion ,Tfno]

print("Los datos personales son: " + listaDatos[0] + " " +
      listaDatos[1] + " " + listaDatos[2])
```

6. CONDICIONALES

SOLUCIÓN EJERCICIO 3

```
num1=int(input("Introduce el primer número: "))
num2=int(input("Introduce el segundo número: "))
num3=int(input("Introduce el tercer número: "))
media=(num1+num2+num3)/3

print("La media aritmética es: ")
print(media)
```


6. CONDICIONALES

CONDICIONALES – CONCATENACIÓN DE COMPARADORES

```
print("Verificacion de edad")  
edad=-7  
  
if 0<edad<100:  
    print("Edad correcta")  
else:  
    print("Edad incorrecta")
```

```
salario_presidente=int(input("Introduce el salario del presidente"))  
print("Salarios: " + str(salario_presidente))  
  
salario_director=int(input("Introduce el salario del director"))  
print("Salarios: " + str(salario_director))  
  
salario_jefe_area=int(input("Introduce el salario del jefe de area"))  
print("Salarios: " + str(salario_jefe_area))  
  
salario_administrativo=int(input("Introduce el salario del administrativo"))  
print("Salarios: " + str(salario_administrativo))  
  
if salario_administrativo<salario_jefe_area<salario_director<salario_presidente:  
    print("Todo funciona correctamente")  
else:  
    print("Algo falla en esta empresa")
```

6. CONDICIONALES

OPERADORES AND Y OR

EJERCICIO.

Para obtener una beca se necesita cumplir los siguientes requisitos:

- Vivir a una distancia > 40 Km
- Numero de hermanos > 2
- Salario familiar <= 20000

```
print("Programa de becas")
distancia_escuela=int(input("introduce la distancia "))
print(distancia_escuela)

numero_hermanos=int(input("Introduce numero hermanos "))
print(numero_hermanos)

salario_familiar=int(input("Introduce salario familiar "))
print(salario_familiar)

#vamos a valorar las 3 variables a la vez

if distancia_escuela>40 and numero_hermanos>2 and salario_familiar<=2000:
    print("Tienes derecho a beca")
else:
    print("No tienes derecho a beca")
```

6. CONDICIONALES

OPERADOR IN

EJERCICIO.

Un alumno tiene que escoger una asignatura optativa. Se le van a ofrecer un listado de asignaturas. Si escoge una asignatura del listado, se matriculará correctamente. Si no, el programa le dirá que no puede matricularse.

```
print("Escoge asignatura")
print("1. PIO")
print("2. INGLES")
print("3. ETC")

asignatura=int(input("Escoge el numero de asignatura: "))

if asignatura in (1, 2, 3):
    print("Matriculado en la asignatura " + str(asignatura))
else:
    print("Ha escogido una asignatura incorrecta")
```

7. BUCLES

7.1. BUCLE FOR

for variable in elemento a recorrer:

LISTA, TUPLA,
CADENA DE TEXTO...

```
for i in [1,2,3]:  
    print("Hola")  
  
print("")  
  
for j in ["primavera","verano","otoño","invierno"]:  
    print("Hola")
```

Ambos for ejecutan
el print tantas veces
como elementos hay
dentro de la lista

```
for i in [1,2,3]:  
    print(i)  
  
print("")  
  
for j in ["primavera","verano","otoño","invierno"]:  
    print(j)
```

Ahora imprimen los
elementos que hay
dentro de las listas

7. BUCLES

7.1. BUCLE FOR

Es posible evitar que cada impresión se haga en una línea diferente.

```
for i in ["Ana", "Paco", "Sonia"]:  
    print("Hola", end=" ")
```

End evita que cada
print se haga en una
línea diferente

```
for i in "sorenes@ifp.es"  
    print("Hola", end=" ")
```

Imprime Hola tantas veces
como letras hay en la
dirección de email

7. BUCLES

7.1. BUCLE FOR

EJERCICIO. Comprueba si una dirección de correo es correcta o no en función de si tiene o no el símbolo @ y un .

```
contador=0
miEmail=input("Introduce tu direccion de email: ")

for i in miEmail
    if i=="@" or i=="."
        contador = contador + 1

if contador==2:
    print("El email es correcto")
else:
    print("El email no es correcto")
```

7. BUCLES

7.1. BUCLE FOR

```
for i in range(5):  
    print("Hola")
```

Range crea una especie de array,
en este caso de 5 elementos

```
for i in range(5):  
    print(i)
```

El resultado será 0, 1, 2, 3, 4

EJERCICIOS PROPUESTOS

1. Define una función llamada `num_max_min()` que nos devuelva en pantalla el número mayor y menor entre 3 diferentes enteros. En caso de que todos sean iguales imprime en pantalla un mensaje indicándolo.
2. Define una función que permita imprimir un mensaje en base a los valores tomados de una lista para comprobar si todos los de la lista son mayores o menores de edad.
3. Define una función que permita multiplicar los números de una lista y mostrar el resultado.
4. Define una función que pida dos números enteros y que escriba si el mayor es múltiplo del menor

EJERCICIOS PROPUESTOS

SOLUCIÓN EJERCICIO 1.

```
def num_max_min(a, b, c):  
    if a > b and a > c:  
        print ("El mayor es", a, "y el menor", c)  
    elif b > a and b > c:  
        print ("El mayor es", b, "y el menor", c)  
    elif c > a and c > b:  
        print ("El mayor es", c, "y el menor", b)  
    else:  
        print ("Son iguales")  
num_max_min(4,2,1)
```

EJERCICIOS PROPUESTOS

SOLUCIÓN EJERCICIO 2.

```
def mayor_menor_edad (lista):  
    for i in lista:  
        if i > 18:  
            print ("Es mayor de edad")  
        elif i == 18:  
            print ("Apenas tiene la mayoría de edad")  
        else:  
            print ("Es menor de edad")  
mayor_menor_edad([18,21,8,19,5,4,3,8,2,3])
```

EJERCICIOS PROPUESTOS

SOLUCIÓN EJERCICIO 3

```
def multip (lista):  
    multiplicacion = 1  
    for i in lista:  
        multiplicacion *= i  
    print (multiplicacion)  
multip([4,2,6])
```

EJERCICIOS PROPUESTOS

SOLUCIÓN EJERCICIO 4

```
def multiplos():  
    print("COMPARADOR DE MÚLTIPLOS")  
    numero_1 = int(input("Escriba un número: "))  
    numero_2 = int(input("Escriba otro número: "))  
  
    if numero_1 >= numero_2 and numero_1 % numero_2 != 0:  
        print(str(numero_1) + " no es múltiplo de " + str(numero_2))  
    elif numero_1 >= numero_2 and numero_1 % numero_2 == 0:  
        print(str(numero_1) + " es múltiplo de " + str(numero_2))  
    elif numero_1 < numero_2 and numero_2 % numero_1 != 0:  
        print(str(numero_2) + " no es múltiplo de " + str(numero_1))  
    else:  
        print(str(numero_2) + " es múltiplo de " + str(numero_2))  
  
multiplos()
```

7. BUCLES

7.1. BUCLE WHILE

While condición:

Cuerpo del bucle

```
i=1

while i<=10
    print("Ejecución " + str(i))
    i = i+1

print("Terminó la ejecución")
```

7. BUCLES

7.1. BUCLE WHILE – Actividad propuesta

Escriba un algoritmo que sume los números ingresados por el usuario y cuando la suma sea superior a 100 o el usuario introduzca un 0, deje de pedir números y muestre el total.

```
suma = 0
numero = int(input(u"Ingrese un número"))
while numero!=0:
    suma += numero
    if (suma>100):
        break
    numero = int(input(u"Ingrese un número"))
print u"Suma total:",suma
```

7. BUCLES

7.1. BUCLE WHILE – Actividad propuesta

2. Crea una tupla con los meses del año, pide números al usuario, si el numero esta entre 1 y la longitud máxima de la tupla, muestra el contenido de esa posición sino muestra un mensaje de error. El programa termina cuando el usuario introduce un cero.

```
meses = ("Enero", "Febrero", "Marzo", "Abril", "Mayo", "Junio", "Julio", "Agosto", "Septiembre", "Octubre",
salir = False
while(not salir):
    numero = int(input("Dame un numero: "))
    if(numero==0):
        salir= True
    else:
        if(numero>=1 and numero<=len(meses)):
            print(meses[numero-1])
        else:
            print("Inserta un numero entre 1 y ",len(meses))
```

7. BUCLES

7.3. CONTINUE Y ELSE

CONTINUE: pasa a la siguiente iteración de bucle.

ELSE: funciona de forma similar a como la hace dentro de un IF.

```
for letra in "Python":  
    if letra=="h":  
        continue  
    print("Viendo la letra: " + letra)
```


7. BUCLES

7.3. CONTINUE Y ELSE

Queremos contar el número de letras (sin espacios) que tiene un String

```
nombre="Sistemas de gestion empresarial"
contador=0
for i in nombre:
    if i==" ":
        continue
    contador+=1
print(contador)
```

7. BUCLES

7.3. CONTINUE Y ELSE

Queremos comprobar si un correo electrónico tiene una @

```
email=input("Introduce tu email, por favor: ")
for i in email:
    if i=="@":
        arroba=True
        break;
    else:
        arroba=False
print(arroba)
```

Forma parte del bucle for.

Se ejecuta cuando acaba el bucle, a no ser que se salga del bucle de forma forzada.



*La mejor forma de aprender es “haciendo”.
Educación y empresas forman un binomio inseparable*