

SISTEMAS DE GESTIÓN EMPRESARIAL

Curso: 2º Desarrollo de Aplicaciones Multiplataforma

Centro: Escuela Arte Granada

Profesor: Daniel López Lozano





TEMA 3.

Introducción al Lenguaje Python

INDICE DE CONTENIDOS



1) Historia de Python

2) Variables y tipos de
datos

3) El tipo String

4) Módulos de Python

5) Condicionales if

6) Bucle while

7) Listas y Tuplas

8) Bucle for

9) Diccionarios

10) Funciones

HISTORIA DE PYTHON



INTRODUCCIÓN A PYTHON

- ❑ Python es un lenguaje de programación potente y accesible de aprender.
- ❑ Posee eficientes estructuras de datos de alto nivel y un enfoque sencillo, pero efectivo de la programación orientada a objetos.
- ❑ La sintaxis elegante, los tipos de datos dinámicos y el hecho de ser un lenguaje interpretado hacen de Python un lenguaje ideal para la creación de scripts (automatizar tareas rutinarias) y el desarrollo rápido de aplicaciones en todo tipo de áreas y plataformas.

INTRODUCCIÓN A PYTHON



- ❑ Creado por Guido van Rossum en 1990.
- ❑ El nombre "Python" viene dado por la afición de Van Rossum al grupo Monty Python
- ❑ Decidió empezar el proyecto como un pasatiempo dándole continuidad al lenguaje de programación ABC
- ❑ En la comunidad de Python se le conoce como *Benevolente Dictador Vitalicio*

CARACTERISTICAS

PYTHON 3

Es un lenguaje pensado para
hacer las cosas simples

Multiparadigma

Procedimental, POO y funcional

Multiproposito

Servidores web, juegos, IA,
ciencia de datos

Interpretado

Tipado dinámico y multiplataforma

VENTAJAS

- ❑ Fácil de aprender
- ❑ Software libre
- ❑ Alta productividad
- ❑ Librería estándar muy extensa y optimizada
- ❑ Uno de los lenguajes más usados y en constante crecimiento

Aug 2022	Programming Language		Ratings
1		Python	15.42%
2		C	14.59%
3		Java	12.40%
4		C++	10.17%

INCOVENIENTES

- ❑ Rendimiento. Comparado con otros lenguajes como C o C++
 - ❑ Lentitud. Debido principalmente a su naturaleza dinámica
 - ❑ Consumo alto de memoria. También debido a lo anterior
- ❑ Casi nula presencia en al programación para dispositivos móviles

ZEN DE PYTHON

- ❑ Bello es mejor que feo.
- ❑ Explícito es mejor que implícito.
- ❑ Simple es mejor que complejo.
- ❑ Complejo es mejor que complicado.
- ❑ Plano es mejor que anidado.
- ❑ Espaciado es mejor que denso.
- ❑ La legibilidad es importante.

ZEN DE PYTHON

- ❑ Los casos especiales no son lo suficientemente especiales como para romper las reglas.
- ❑ Sin embargo la practicidad le gana a la pureza.
- ❑ Los errores nunca deberían pasar silenciosamente.
- ❑ A menos que se silencien explícitamente.
- ❑ Frente a la ambigüedad, evitar la tentación de adivinar.

ZEN DE PYTHON

- ❑ Debería haber una, y preferiblemente solo una, manera obvia de hacerlo.
- ❑ A pesar de que esa manera no sea obvia a menos que seas Holandés.
- ❑ Ahora es mejor que nunca. A pesar de que nunca es muchas veces mejor que **ahora** mismo.
- ❑ Si la implementación es difícil de explicar, es una mala idea.

ZEN DE PYTHON

- ❑ Si la implementación es fácil de explicar, puede que sea una buena idea.
- ❑ Los espacios de nombres son una gran idea, ¡tenemos más de esos!



VARIABLES Y TIPOS DE DATOS



VARIABLES Y TIPOS DE DATOS

Tipos básicos

Se asigna el tipo en función del valor que recibe la variable

Entero

Números enteros de toda la vida

Float

Números reales

Boolean

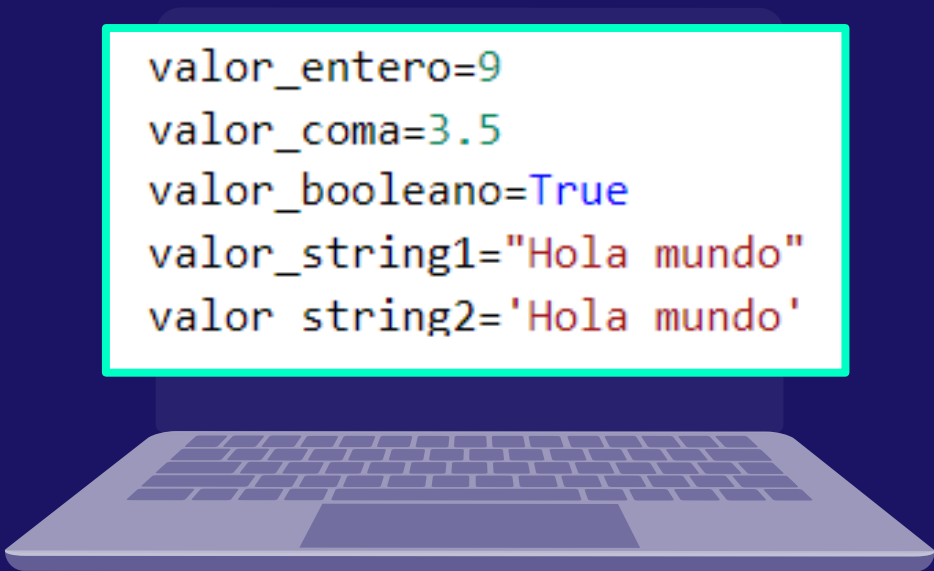
True y False

String

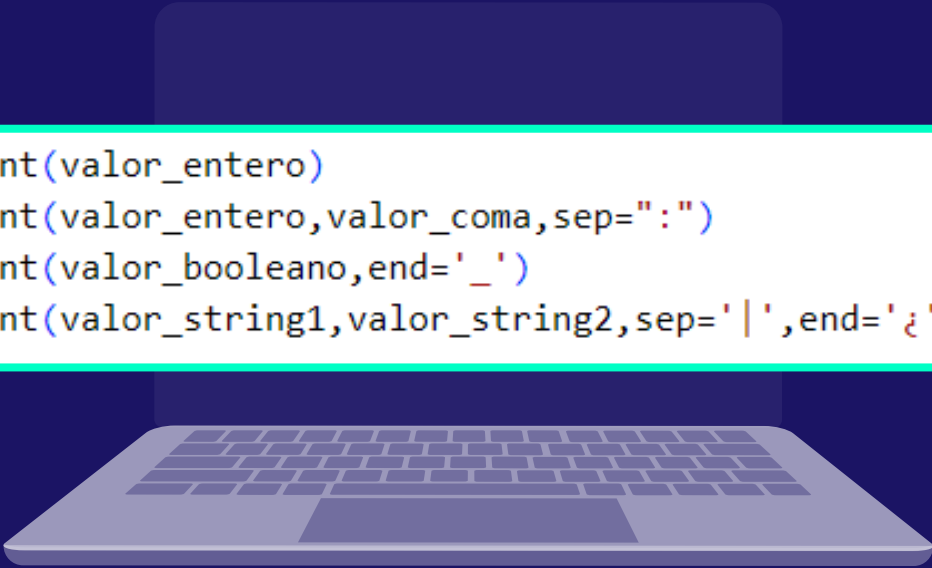
Se crean con comillas simple o dobles

DEFINIR VARIABLES

- ❑ El tipo char o carácter no existe explícitamente, es un string de un solo carácter.
- ❑ Usar comillas simples o dobles para los string es igualmente válido



```
valor_entero=9  
valor_coma=3.5  
valor_booleano=True  
valor_string1="Hola mundo"  
valor_string2='Hola mundo'
```

```
print(valor_entero)
print(valor_entero,valor_coma,sep=":")
print(valor_booleano,end='_')
print(valor_string1,valor_string2,sep='|',end=';')
```

print

- ❑ Función que permite escribir datos en la salida estándar (consola)
- ❑ sep y end son dos parámetros especiales que establecen que se usa de separador y fin de línea
- ❑ ¿Para que puede servir cambiar end?

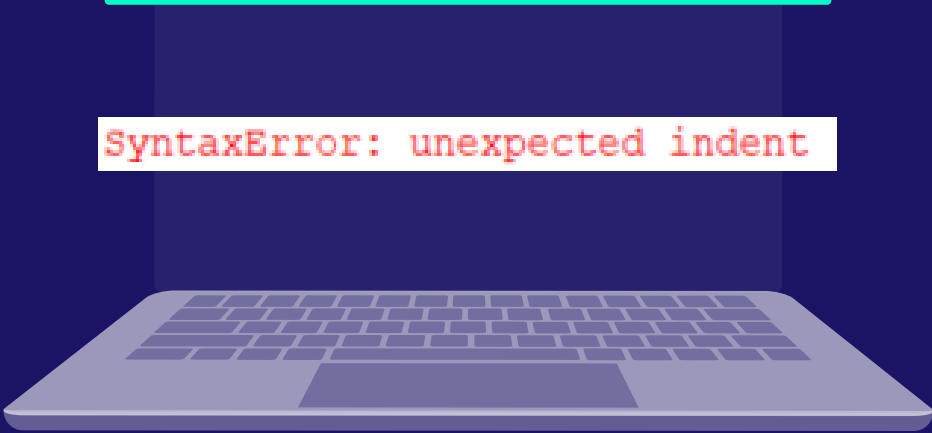
type

- ❑ Los tipos de datos que existen en el lenguaje Python se pueden obtener como un string con su información

```
print(type(valor_entero),type(valor_string1))
```

```
<class 'int'> <class 'str'>
```





```
nombre="Juan Pedro"  
print(nombre)
```

```
SyntaxError: unexpected indent
```

ESPACIADO E INDENTACIÓN

- ❑ Los espacios son importantes ya que la indentación puede interpretar el código de manera errónea.

VARIAS LINEAS

- ❑ Python no obliga a poner ; porque se usa para poner varias instrucciones en la misma
- ❑ Si el proceso es al contrario Python no lo admitirá a menos que usemos paréntesis o el carácter \

```
valor_entero=9;valor_coma=3.5;valor_booleano=True

PI=3.14159242
radio = 5
area = PI * radio ** 2
area = (PI * radio
      ** 2)
area = PI * radio \
      ** 2
```





```
#Esto es un comentario
```

```
"""
```

```
Esto funciona como un comentario de varias de lineas  
aunque en realidad se utiliza para escribir  
de varias lineas facilmente
```

```
"""
```

COMENTARIOS

- ❑ Esto se definió para que programa tipo Javadoc o Doxygen generen la documentación de programas

MULTI LINEA


- ❑ Cuando tengamos cadenas de caracteres muy grandes y queremos ver todo el código por pantalla
- ❑ No importa el espaciado dentro de estas comillas y se comportan de manera equivalente a las otras formas de crear un string

```
print("""  
    Esto funciona como un comentario de varias de lineas  
    aunque en realidad se utiliza para escribir  
    de varias lineas facilmente  
""")  
  
texto="""  
    Esto funciona como un comentario de varias de lineas  
    aunque en realidad se utiliza para escribir  
    de varias lineas facilmente  
    """
```



OPERACIONES ARITMETICAS

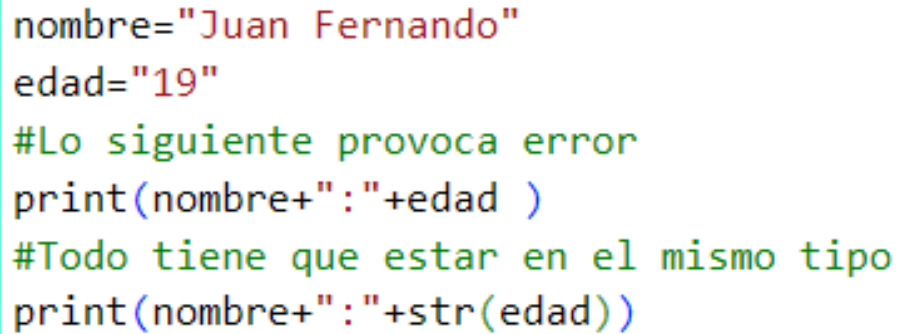
$A + B$	Suma
$A - B$	Resta
$A * B$	Multiplicación
$A \% B$	Resto
A / B	División real
$A // B$	División entera
$A ** B$	Potencia



```
a,b=1,4    #Asignacion multiple
a,b="Hola","Mundo"
a,b=b,a    #Intercambia las variables
num=1
num+=2
num-=1
x=None     #Valor no definido
```

ATAJOS DE CÓDIGO

- ❑ Hay muchos más en Python ya que es famoso por tener ciertas expresiones que atajan estructuras más o menos complejas
- ❑ El valor **None** es similar al **null** pero no solo para variables de tipo objeto



```
nombre="Juan Fernando"
edad="19"
#Lo siguiente provoca error
print(nombre+": "+edad )
#Todo tiene que estar en el mismo tipo
print(nombre+": "+str(edad))
```

COMPATIBILIDAD DE TIPOS

- ❑ En una expresión Python los tipos tienen que concordar ya que no se realizan conversiones implícitas entre tipo no relacionados como por ejemplo entre string y enteros

CONVERSIONES DE TIPOS

```
>>> str(23)
'23'
>>> str(3.5)
'3.5'
>>> str(True)
'True'
>>> float(12)
12.0
>>> float('13.65')
13.65
>>> int('123')
123
>>> int(13.54)
13
>>> bool(0)
False
>>> bool('')
False
>>> bool('Hola')
True
>>> bool(-3.14)
True
```

AJUSTE DE DECIMALES

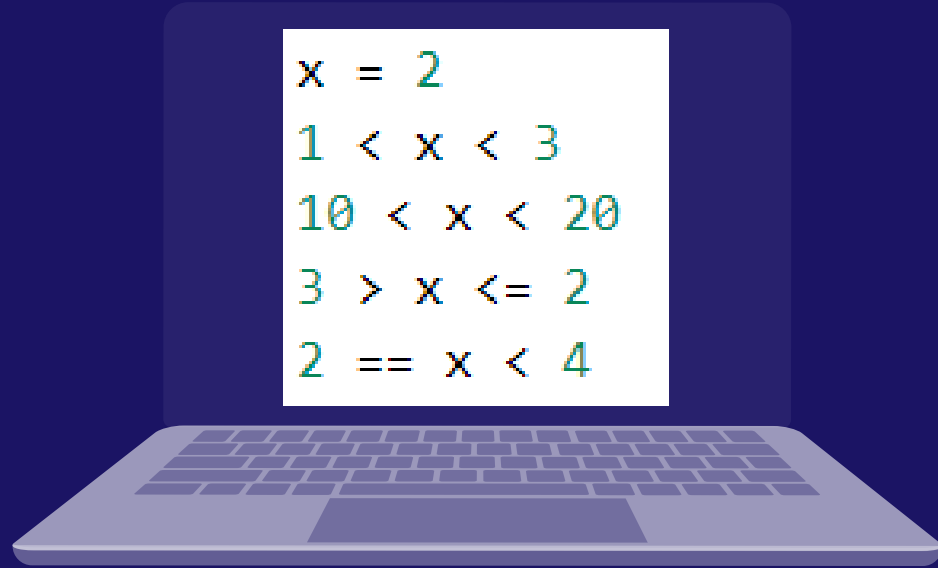
```
>>> num1 = 1.56234
>>> num2 = 1.434
>>> print(round(num1))
2
>>> print(round(num2))
1
>>> print(round(num1, 3))
1.562
>>> print(round(num2, 1))
1.4
```

TEMPLATES

- ❑ Si queremos mezclar string con variables de distintos tipo lo mejor es usar templates o plantillas
- ❑ Incluso se puede realizar operaciones de cualquier tipo dentro de un template

```
nombre = "Pepe"  
edad = 25  
print(f"Me llamo {nombre} y tengo {edad} años.")  
  
semanas = 4  
print(f"En {semanas} semanas hay {7 * semanas} días.")
```





```
x = 2
```

```
1 < x < 3
```

```
10 < x < 20
```

```
3 > x <= 2
```

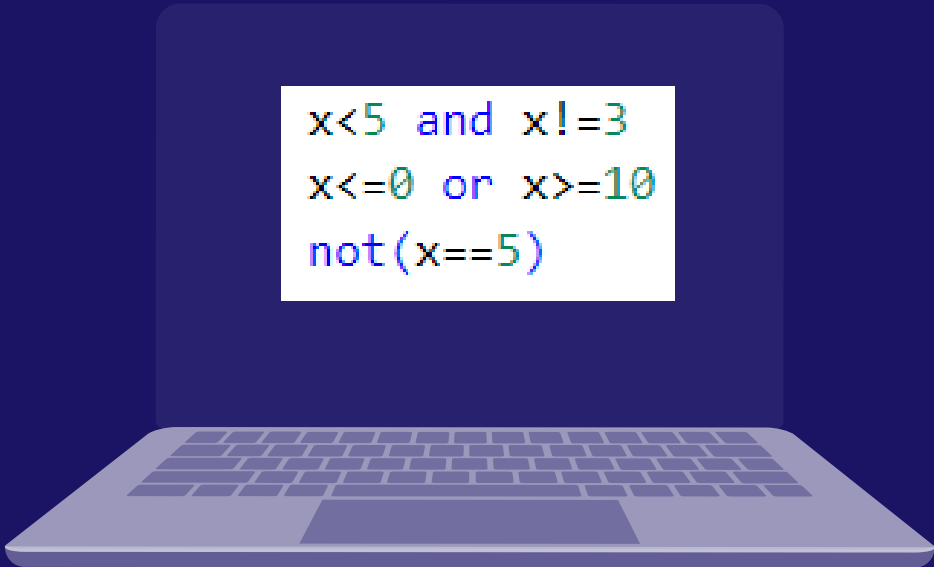
```
2 == x < 4
```

OPERADORES RELACIONALES

- ❑ Operaciones de comparación que devuelven un valor True o False
- ❑ Python permite combinar en una sola expresión varios operadores relacionales

OPERADORES BOOLEANOS

- ❑ Se escriben literalmente como se pronuncian en inglés: **and** **or** **not**



```
x<5 and x!=3  
x<=0 or x>=10  
not(x==5)
```

ENTRADA DE DATOS

- ❑ La entrada estándar (teclado) se obtiene mediante la función `input`.
- ❑ Devuelve siempre un string por lo que hay que convertirlo explícitamente al tipo necesario en cada momento

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

radio = float(input("Radio de la circunferencia: "))
print(f"La circunferencia con {radio} de radio tiene {PI * radio ** 2}")
```

EJERCICIOS PROPUESTOS

EJERCICIO 1

Escribir un programa que pida al usuario su peso (en kg) y estatura (en metros), calcule el índice de masa corporal redondeado a 2 decimales.

EJERCICIO 2

Escribir un programa que pida una cantidad de días y las convierta a horas minutos y segundos. Usar templates para mostrar los datos.

EJERCICIO 3

Escriba un programa conversor de centímetros a kens y shakus, unidades japonesas de longitud. Un ken son seis shakus y un shaku equivale a 30,3 cm. Se pedirá una cantidad de centímetros y mostrar los resultados con 2 decimales.

EJERCICIO 4

Investigar como escribir un programa que transforme un número expresado en binario como un string, a base decimal sin bucles usando la función int.

EL TIPO STRING



ACCESO A UN STRING

- ❑ Cada carácter tiene asociado un índice que permite acceder a él
- ❑ `cadena[i]` devuelve el carácter de la cadena en el índice `i`

Cadena	P	y	t	h	o	n
Índice positivo	0	1	2	3	4	5
Índice negativo	-6	-5	-4	-3	-2	-1

```
lenguaje='Python'
```

```
lenguaje[0] #-> P
```

```
lenguaje[1] #-> y
```

```
lenguaje[-1] #-> n
```

```
lenguaje[6]
```

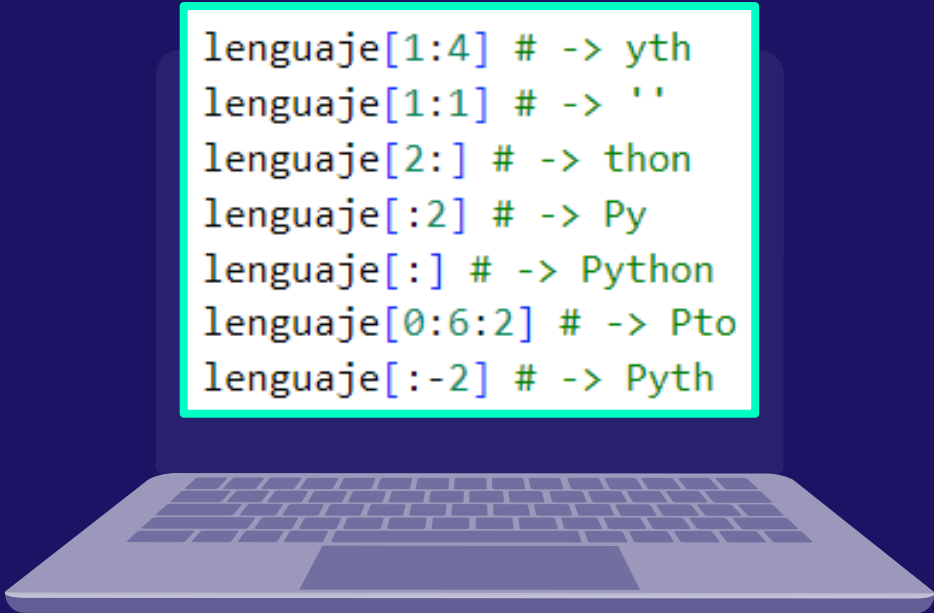
```
Traceback (most recent call last):  
  File "<pyshell#8>", line 1, in <module>  
    lenguaje[6]  
IndexError: string index out of range
```

POSICIONES STRING

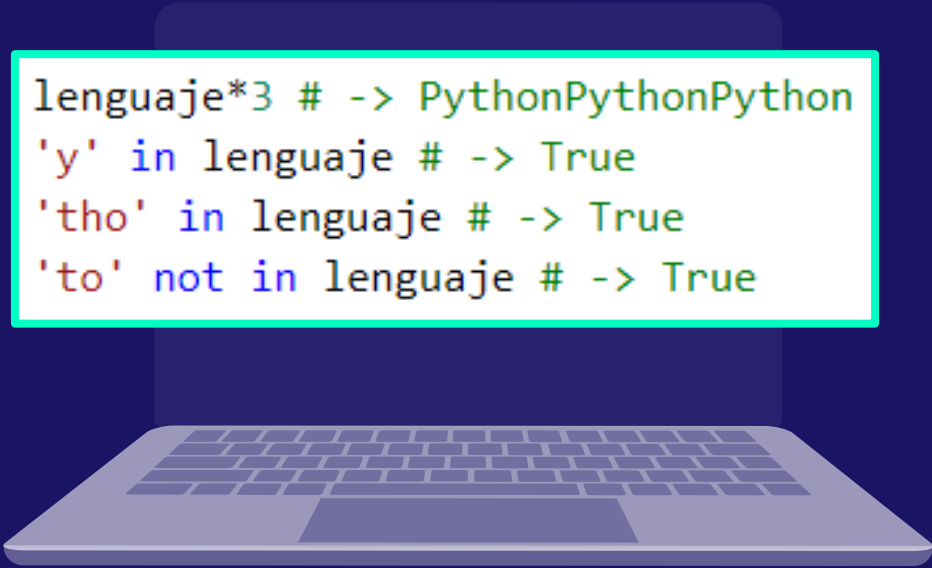
- ❑ Admite posiciones negativas, pero no por encima del tamaño de la cadena

SUBCADENAS

- ❑ `cadena[i : j : k]` devuelve la subcadena desde el carácter con el índice `i` hasta el carácter anterior al índice `j` tomando caracteres cada `k`



```
lenguaje[1:4] # -> yth  
lenguaje[1:1] # -> ''  
lenguaje[2:] # -> thon  
lenguaje[:2] # -> Py  
lenguaje[:] # -> Python  
lenguaje[0:6:2] # -> Pto  
lenguaje[:-2] # -> Pyth
```



```
lenguaje*3 # -> PythonPythonPython
'y' in lenguaje # -> True
'tho' in lenguaje # -> True
'to' not in lenguaje # -> True
```

MÁS OPERACIONES

- ❑ * repite la cadena N veces
- ❑ in y not in nos dice si contiene subcadenas o no
- ❑ Las operaciones y métodos sobre cadenas devuelve valores nunca modifican la cadena (inmutables)

COMPARACION DE CADENAS

- ❑ $C1==C2$ devuelve True si las cadenas son iguales
- ❑ $C1!=C2$ devuelve True si las cadenas son distintas
- ❑ $C1>C2$ devuelve True si la cadena C1 sucede a la C2
- ❑ $C1<C2$ devuelve True si la cadena C1 precede a la C2
- ❑ $C1>=C2$ devuelve True si la cadena C1 sucede a la C2 o son iguales
- ❑ $C1<=C2$ devuelve True si la cadena C1 precede a la C2 o son iguales

COMPARACIONES

```
lenguaje == 'python' # -> False  
lenguaje < 'python' # -> True  
' ' < lenguaje # -> True
```



METODOS DE LA CLASE STRING

```
len(lenguaje) # -> 6
lenguaje.upper() #Devuelve PYTHON (immutable)
dato='123'
dato.isnumeric() # -> true
dato.isalpha() # -> false
dato.isalnum() # -> false
entrada='  Hola  '
entrada.strip() # -> 'Hola'
palabra='pepipo'
palabra.islower() # -> True
palabra.find('e') # -> 1
palabra.find('=') # -> -1
palabra.find('pi') # -> 2
palabra.count('p') # -> 3
palabra.replace('p','n') # -> nenino
palabra.replace('p','n',2) # -> nenipo
dir = "C:/python36/python.exe"
dir.find("/") # Retorna la primera ocurrencia -> 2
dir.rfind("/") # Retorna la última ->11
```


METODOS DE LA CLASE STRING

- ❑ Hay más métodos interesantes, pero sería necesario conocer las listas
- ❑ Hay muchas otras que solo son útiles en casos muy concretos

<https://recursospython.com/guias-y-manuales/30-metodos-de-las-cadenas/>

EJERCICIOS PROPUESTOS

EJERCICIO 5

Escriba un programa que pida un string y lo muestre al revés sin usar bucles.

EJERCICIO 6

Escriba un programa que pida un numero y nos diga si es un float (mostrar True o False).

EJERCICIO 7

Escriba un programa que pida un string y cree una variable que esté formada por los dos primeros y por los dos últimos caracteres. Suponemos que tiene la palabra al menos 4 caracteres y hay que resolverlo con subcadenas.

EJERCICIO 8

Escriba un programa que pida un string y cree una variable que esté con el mismo string pero con el primero y último carácter intercambiados. Puede tener cualquier longitud. Resolverlo con subcadenas.

MÓDULOS DE PYTHON



LA BIBLIOTECA ESTANDAR

- ❑ Python contiene unas de las librerías estándar más completas (build-in-functions)
- ❑ En un futuro veremos que podemos crear nuestros propios módulos importables de manera muy sencilla e importar los de terceros.
- ❑ Esto, entre otras cosas, es lo que ha hecho crecer el lenguaje hasta lo popular que es hoy, ya que el lenguaje se extiende sin parar.

IMPORTACION COMPLETA

- ❑ Si usamos la declaración `import` estamos incluyendo todo el modulo al completo.
- ❑ Además de tener que poner el prefijo del nombre del módulo delante para poder usar el objeto/función concreta

```
import datetime
import random
import math
import os.path

os.getcwd()
os.path.exists("C:/python36/python.exe")
math.pi
math.gcd(12, 16)
math.factorial(5)
random.random()
random.randint(1, 5)
datetime.datetime.now().date()
datetime.datetime.now().time()
```

```
from datetime import datetime
from random import random, randint
from math import pi, gcd, factorial
from os import getcwd
from os.path import exists
```

```
getcwd()
exists("C:/python36/python.exe")
pi
gcd(12, 16)
factorial(5)
random()
randint(1, 5)
datetime.now().date()
datetime.now().time()
```

IMPORTACION SELECTIVA

- ❑ Usando from seremos más precisos y eficientes, además de que ya no tenemos que poner de prefijo el nombre del módulo.
- ❑ Será nuestra forma habitual

CONDICIONALES

(if-elif-else)



DEFINICIÓN DE CONDICIONAL

```
numero = int(input("Escriba un número positivo: "))  
if numero < 0:  
    print("¡Le he dicho que escriba un número positivo!")  
print(f"Ha escrito el número {numero}")
```

```
edad = int(input("¿Cuántos años tiene? "))  
if edad < 18:  
    print("Es usted menor de edad")  
else:  
    print("Es usted mayor de edad")  
print("¡Hasta la próxima!")
```

```
edad = int(input("¿Cuántos años tiene? "))  
if edad < 0:  
    print("No se puede tener una edad negativa")  
elif edad >= 0 and edad < 18:  
    print("Es usted menor de edad")  
else:  
    print("Es usted mayor de edad")
```


DEFINICIÓN DE CONDICIONAL

- ❑ La estructura de control if ... permite que un programa ejecute unas instrucciones cuando se cumplan una condición.
- ❑ Pueden aparecer varios bloques elif pero solo un else al final.
- ❑ Los bloques de código se diferencia por estar identados 4 espacios (o más) ocurriendo para cualquier otra estructura definida en bloques

EJEMPLOS DE CONDICIONALES

- ❑ La indentación puede ser de más de 4 espacios siempre que todo el bloque la respete.
- ❑ Los otros bloques son independientes del espaciado usado en otros bloques

```
edad = int(input("¿Cuántos años tiene? "))
if edad < 18:
    print("Es usted menor de edad")
    print("Recuerde que está en la edad de aprender")
else:
    print("Es usted mayor de edad")
    print("Recuerde que debe seguir aprendiendo")
print("¡Hasta la próxima!")
```

EJEMPLOS DE CONDICIONALES

- ❑ La instrucción `pass` permite indicar que un bloque de código está vacío.
- ❑ Es necesario al no haber nada que indique el comienzo y fin de un bloque.
- ❑ Esto no es exclusivo de los `if` ya que comúnmente se utiliza al definir un método o función

```
edad = int(input("¿Cuántos años tiene? "))
if edad < 120:
    pass
else:
    print("¡No me lo creo!")
print(f"Usted dice que tiene {edad} años.")
```

CONDICIONALES ANIDADOS

```
print("Este programa mezcla dos colores.")
print("  r. Rojo      a. Azul")
primera = input("  Elija un color (r o a): ")
if primera.lower() == "r":
    print("  a. Azul      v. Verde")
    segunda = input("  Elija otro color (a o v): ")
    if segunda.lower() == "a":
        print("La mezcla de Rojo y Azul produce Magenta.")
    else:
        print("La mezcla Rojo y Verde produce Amarillo.")
else:
    print("  v. Verde      r. Rojo")
    segunda = input("  Elija otro color (v o r): ")
    if segunda.lower() == "v":
        print("La mezcla de Azul y Verde produce Cian.")
    else:
        print("La mezcla Azul y Rojo produce Magenta.")
print("¡Hasta la próxima!")
```

EJEMPLOS DE CONDICIONALES

- ❑ Si la expresión implica a números y depende de que el valor resultante sea 0 no es necesario introducir el operador de comparación

```
numero = int(input("Escriba un número: "))  
if numero % 2:  
    print(f"{numero} es impar")  
else:  
    print(f"{numero} es par")
```

EJERCICIOS PROPUESTOS

EJERCICIO 9

Escribir un programa que pida un número float y calcule su potencia de exponente entero que también se pide por teclado. Si alguna entrada de datos no es incorrecta mostrar un mensaje de error.

EJERCICIO 10

Escribir un programa que simule el juego de piedra, papel, tijera en la que existen dos jugadores (Ana y Juan). Ambos representados por la máquina generando valores aleatorios de manera que el valor 1 corresponda a piedra, el valor 2 corresponda a papel y el valor 3 corresponda a tijera.

EJERCICIO 11

Escribir un programa que pida el número de golpes que ha necesitado un jugador de golf para hacer hoyo y el par del hoyo que es número ideal de golpes. El programa debe calcular la puntuación del jugador que depende de los valores anterior mediante el siguiente resumen:

EJERCICIOS PROPUESTOS

El programa deberá mostrar el mensaje correspondiente según la puntuación que obtenga el jugador:

```
1           "Hole-in-one!"
<= par - 2  "Eagle"
par - 1     "Birdie"
par         "Par"
par + 1     "Bogey"
par + 2     "Double Bogey"
>= par + 3  "Go Home!"
```

BUCLE while



DEFINICIÓN DE WHILE

- ❑ Repite la ejecución del bloque de código mientras la expresión lógica sea cierta.
- ❑ El bloque de código debe estar indentado por al menos 4 espacios.
- ❑ Es un bucle indefinido porque no se sabe a priori cuantas veces se va a repetir según el código de actualización de las variables que tengan que ver en la condición

EJEMPLOS DE WHILE

```
i = 1
while i <= 100:
    print(i)
    i += 1
print("Programa terminado")
```

```
i = 1
while i <= 50:
    print(i, end=" ")
    i += 2
print("...Programa terminado")
```

```
i = 1
while i <= 100:
    print(i, end=" ")
print("No se escribe nunca")
```

EJERCICIOS PROPUESTOS

EJERCICIO 12

Escribe un programa que pida datos al usuario hasta que escriba una cadena alfanumérica.

EJERCICIO 13

Escribe un programa que pida un numero por pantalla y usando un solo while escriba un triangulo y un cuadrado de asteriscos con el tamaño indicado.

EJERCICIO 14

Escribe un programa que genere un número aleatorio entre 1 y 50. Después el usuario intentará acertarlo.

EJERCICIO 15

Escribe un programa similar al anterior, pero donde el propio programa sea el que intenta acertar un número que piensa el usuario generando aleatorios. El usuario responde s o n hasta que el programa acierta el número. Hacerlo de manera lo más inteligente posible y no solo con números al azar en el rango 1-50.

LISTAS Y TUPLAS



DEFINICIÓN DE LISTA

- ❑ Es una estructura de datos utilizada para almacenar múltiples valores en secuencia
- ❑ Los valores se almacena ordenados
- ❑ Pueden contener valores de cualquier tipo y una misma lista contener valores de varios tipos
- ❑ Cada posición en la lista está asociada a un entero llamado índice.
- ❑ Es mutable, es decir, puede ser modificada tanto en contenido como en tamaño

EJEMPLOS DE LISTAS

```
vacia=[]  
primos = [2, 3, 5, 7, 11, 13]  
  
diasLaborables = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes"]  
  
peliculas = [["Senderos de Gloria", 1957], ["Hannah y sus hermanas", 1986]]  
  
matriz=[['a','b','c','d'],  
        [1,2,3,4],  
        ["Cinco","Seis","Siete"]]  
  
tuti_fruti=[1, "balanza", 45, True]
```

CREAR LISTAS

- ❑ Podemos crear listas a través de otras variables.
- ❑ De la misma forma podemos desestructurar la lista en sus posiciones constituyentes

```
nombre = "Pepe"  
edad = 25  
persona = [nombre, edad]  
[name, age] = persona #name es Pepe y age es 25
```



```
palabras=['alto','claro','barca','tren']
palabras[0] # ->'alto'
palabras[2] # ->'barca'
palabras[6] # -> Error se corta el programa
palabras[-1] # ->'tren'
#Modificar el valor de una posicion
palabras[1]='oscuro'
```



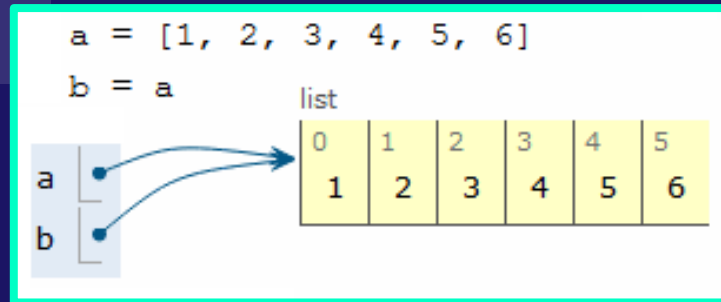
ACCEDER Y MODIFICAR

- ❑ Usando enteros de la misma manera que los string (se puede usar negativos)
- ❑ Al acceder por posición no dejan de ser variables por lo que mediante asignación los modificamos



REFERENCIA Y COPIA DE LISTAS

- ❑ Las listas entre otros objetos en Python se manejan por referencia.
- ❑ Esto significa que si usamos la asignación entre dos listas no estamos copiando una en otra, sino que tenemos dos referencias a la misma lista




POR REFERENCIA POR VALOR

- ❑ La copia por referencia implica que que si modificamos cualquier lista la otra también ser modificara ya que realmente son la misma.
- ❑ Con la función `list` hacemos una copia nueva (clon) de la misma lista

```
numeros=[1,3,4,6,7]
copia=numeros
numeros[1]=13
#copia cambia tambien con esto

copia=list(numeros)
#copia y primos son independientes
```





```
palabras[1:4] # -> ['claro','barca','tren']
palabras[1:1] # -> []
palabras[:3]  # -> ['alto','claro','barca']
palabras[0:4:2] # -> ['claro','tren']
palabras[0::2] # -> ['claro','tren']
```

SUBLISTAS


- ❑ Funcionan igual que las subcadenas de string
- ❑ El resultado que se obtiene es una lista nueva (copia) con los datos seleccionados

UNION DE LISTAS

- ❑ El operador + concatena lista en el orden en que se establezcan
- ❑ Se crea una nueva lista
- ❑ En el ultimo ejemplo se crea una lista nueva en la misma variable, OJO CUIDAO, la lista no muta

```
a = [1,3]
b = [2,4,6]
union=a+b # union es una lista nueva
vocales = ["A", "E", "I"]
vocales += ["O","E"]
```





```
len(tuti_fruti) # -> 4
'I' in vocales # -> true
'i' in vocales # -> false

from random import choice
choice(diasLaborables)
# -> cualquiera de los dias
```

OPERACIONES VARIAS


- ❑ Podemos aplicar casi lo mismo que a las cadenas ya que no dejan de ser colecciones también
- ❑ Con las listas interactúan casi todos los módulos de Python de una manera u otro como la random

OPERACIONES VARIAS

- ❑ El método `index` hace fallar el programa si no encuentra el dato.
- ❑ Con el tratamiento de excepciones en el futuro controlaremos

```
vocales.index("0") # -> 3  
vocales.index("o")  
# -> Error se corta el programa  
notas=[1,1,6,5,6,7,8,6]  
notas.count(6) # -> 3  
max(notas) # -> 8  
min(notas) # -> 1
```





```
lenguaje="Python"  
lenguaje.split() # -> ["P","y","t","h","o","n"]  
letras=["P","y","t","h","o","n"]  
"".join(letras) # -> "Python"  
",".join(letras) # -> "P,y,t,h,o,n"  
frase="Vive la vida"  
frase.split(" ") # -> ["Vive","la","vida"]
```

LISTAS Y STRING

- ❑ split permite tratamiento de cadenas para su análisis
- ❑ join permite unir una lista de cadenas

LISTAS A PARTIR DE OTRAS

- ❑ Mediante el operador `*` podemos juntar listas en el orden que queramos.
- ❑ Tengamos en cuenta que la lista resultante es una nueva no vinculada a ninguna de las anteriores
- ❑ Util para programación funcional y comprensión de listas que veremos en el siguiente tema

```
#interesante sobre todo en programacion funcional
lista=[1,2,3]
nueva=[*lista]
# resulta una nueva [1,2,3]

mezcla=[*lista,*[4,5,6]]
# incorporamos datos a una lista y
# resulta una nueva [1,2,3,4,5,6]

otra_lista=[4,5,6]
mezcla=[*lista,*otra_lista]
# resulta una nueva [1,2,3,4,5,6]

otra_mezcla=[*otra_lista,["tomate","lechuga"],*lista]
# resulta una nueva [1,2,3,"tomate","lechuga",4,5,6]
```


METODOS MUTADORES

```
vocales=["a","e"]  
vocales.append("i") # -> ["a","e","i"]  
vocales.extend(["o","u"]) # -> ["a","e","i","o","u"]  
vocales.reverse() # -> ["u","o","i","e","a"]  
  
letra_i=vocales.pop(2) # ->["a","e","o","u"]  
vocales.remove("a") # ->["e","o","u"]  
vocales.insert(0,"a")  
vocales.insert(2,"i")  
vocales.clear() # -> []
```

ORDENACIÓN


- ❑ Mediante sort podemos ordenar los datos de una lista usando su orden predefinido.
- ❑ Mediante reverse se puede cambiar el orden asc o desc
- ❑ La lista se modifica

```
numeros=[4,-7,0,1,3]  
nombres=["Juan","Ana","Rafael","Eustaquia"]
```

```
numeros.sort()  
#[-7,0,1,3,4]  
numeros.sort(reverse=True)  
#[4,3,1,0,-7]
```

```
nombres.sort()  
#["Ana","Eustaquia","Juan","Rafael"]  
nombres.sort(reverse=True)  
#["Rafael","Juan","Eustaquia","Ana"]
```





```
sorted(numeros)
#[-7,0,1,3,4]
sorted(numeros,reverse=True)
#[4,3,1,0,-7]
sorted(nombres)
#["Ana","Eustaquia","Juan","Rafael"]
sorted(nombres,reverse=True)
#["Rafael","Juan","Eustaquia","Ana"]
```

SORTED (NO MUTA)

- ❑ En este caso es una función que recibe los mismos datos que sort y devuelve una nueva lista ordenada
- ❑ Cuando veamos funciones veremos que podemos cambiar el criterio de ordenación

DEFINICIÓN DE TUPLA

- ❑ Una tupla no deja de ser una lista inmutable
- ❑ Los métodos de listas que no modifican se pueden usar también sobre tuplas
- ❑ Son más eficientes que una lista y se usan en los casos donde se necesite una colección y no se vaya a modificar o sean de un solo uso en el bloque donde aparecen
- ❑ Distintos módulos y métodos de Python interactúan mediante tuplas antes que con listas

EJEMPLOS DE TUPLAS

```
vacia=()
primos = (2, 3, 5, 7, 11, 13)

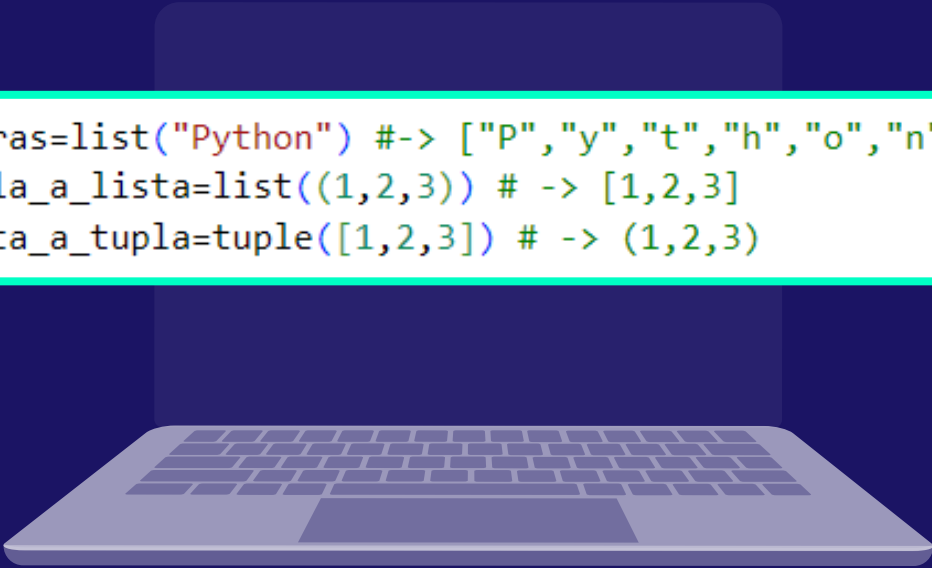
diasLaborables = ("Lunes", "Martes", "Miércoles", "Jueves", "Viernes")

películas = (("Senderos de Gloria", 1957), ("Hannah y sus hermanas", 1986))

matriz=((('a','b','c','d'),
          (1,2,3,4),
          ("Cinco","Seis","Siete"))

tuti_fruti=(1, "balanza", 45, True)

primos[0] # -> 2
primos[1]=12 # -> Error
```



```
letras=list("Python") #-> ["P","y","t","h","o","n"]
tupla_a_lista=list((1,2,3)) # -> [1,2,3]
lista_a_tupla=tuple([1,2,3]) # -> (1,2,3)
```

CONVERSIONES ENTRE COLECCIONES

- ❑ Las listas, tuplas y string (entre otros más) no dejan de ser colecciones
- ❑ Por tanto se puede pasar de unos a otros

EJERCICIOS PROPUESTOS

EJERCICIO 16

Escribir un programa donde un jugador pueda jugar contra la máquina a "piedra", "papel", "tijera" "lagarto" y "spock" de manera que se seleccione un valor entre "piedra", "papel", "tijera" "lagarto" y "spock" por parte del jugador y de manera aleatoria usando choice del módulo random.

EJERCICIO 17

Escribir un programa que dado un texto con mayúsculas y minúsculas separado por espacios indique si es o no un palíndromo.

EJERCICIO 18

Escribir un programa que dado un entero muestre el mismo número, pero dado la vuelta.

EJERCICIO 19

Escribir un programa que dado un entero muestre el mismo número pero con los dígitos ordenados de menor a mayor.

BUCLE for



DEFINICIÓN DE FOR

- ❑ Estructura de control que permite ejecutar uno o varias líneas de código múltiples veces
- ❑ Es el bucle que usamos cuando podemos saber el número de veces que se tiene que repetir un código
- ❑ Tiene una variable de control que se actualiza automáticamente.
- ❑ Es un bucle for inusual ya que se parece mas un for each o bucle iterador sobre una secuencia.

EJEMPLOS DE FOR

- ❑ Si no necesitamos variable ponemos _ de manera que ya no necesitamos inventarnos un nombre

```
print("Comienzo")
for i in [0, 1, 2]:
    print("Hola ", end="")
print("\nFinal")

rango=[1,2,3,4,5] # o una tupla
for i in rango:
    print("Hola ", end="")

for _ in rango:
    print("Hola ", end="")

for i in [3, 4, 5]:
    print(f"Hola. Ahora i vale {i} y su cuadrado {i ** 2}")
```

RANGE

```
pares=0
for i in range(100):
    if i%2:
        pares+=1
print(f"Hay {pares} numeros pares")

for i in range(1, 10, 2):
    print(i, end=" ", " ")
```

- ❑ Si necesitamos rangos más extensos tenemos la función range que genera listas.
- ❑ `range(fin)` : Genera una secuencia desde 0 hasta **fin-1**.
- ❑ `range(inicio, fin, salto)` : Genera una secuencia de números desde **inicio** hasta **fin-1** con un incremento de salto.

RECORRER LISTAS

- ❑ Si no necesitamos la posición de los elementos, un bucle for recorre de manera directa cualquier secuencia

```
datos=["Alba", "Benito", 27,"Antonio",True,'Ventilador',235.90]

for i in range(len(datos)):
    print(datos[i])

for dato in datos:
    print(dato)

vocales=0
for c in "caramelo":
    if c in ['a','e','i','o','u']:
        vocales+=1
print(f"caramelo tiene {vocales} vocales")
```

SUB-SECUENCIAS

- ❑ Usando la notación de sub-listas o sub-cadenas podemos hacer un recorrido versátil y flexible de cualquier colección.

```
for dato in datos[3:6]:  
    print(dato)  
  
for dato in datos[3:]:  
    print(dato)  
  
for dato in datos[:3]:  
    print(dato)  
  
for dato in datos[::-1]:  
    print(dato)
```

EJERCICIOS PROPUESTOS

EJERCICIO 20

Escriba un programa que a partir de dos listas genera listas con los siguientes contenidos:

- Palabras que aparecen en las dos listas.
- Palabras que están en la primera, pero no en la segunda.
- Palabras que aparecen en la segunda, pero no en la primera.
- Palabras que aparecen en ambas listas.

EJERCICIO 21

Escribir un programa que dada una lista elimine sus valores repetidos.

EJERCICIO 22

Escriba un programa que dado un texto nos diga cuantas vocales en total hay en dicho texto.

EJERCICIO 23

Escriba un programa que dado un texto nos muestre el numero de apariciones de cada una de las 5 vocales

DICCIONARIOS



DEFINICIÓN DE DICCIONARIO

- ❑ Es una colección que almacena pares clave-valor
- ❑ La clave sirve para acceder directamente a su valor asociado.
- ❑ Las claves deben ser únicas e inmutables
- ❑ Los valores asociados pueden ser de cualquier tipo
- ❑ Los pares pueden ser modificados, añadidos y eliminados

EJEMPLOS DE DICCIONARIO

```
ejemplo1={'A':4, 'B':5}
#tambien numeros sin un orden concreto
ejemplo1={9:'A', 3:'B', 11:'C'}
trabajador={'nombre':'Alfredo',
            'despacho': 218,
            'email':'asalber@ceu.es',
            'telefono':'123456789'}

tienda={"patatas":20.5,
        "cocacola":2.5,
        "lechuga":5,
        "melon":4.3}
```

ACCESO Y MODIFICACION

- ❑ El método get se usa cuando no sabemos si existe la clave para establecer un valor de respuesta y el programa no colapse
- ❑ Si la clave no existe se añade con la asignación

```
tienda["patatas"] # -> 20.5  
tienda["sandia"] # -> error  
tienda.get("sandia","No existe") # -> No existe  
tienda.get("patatas","No existe") # -> 20.5  
tienda["patatas"]=15 #valor modificado  
tienda["sandia"]=5 #nuevo par clave-valor
```



```
len(tienda) # -> 5 claves-valor
"lechuga" in tienda #-> true
"tinta" in tienda #-> false
min(tienda) # -> cocaCola
max(tienda) # -> sandia

copia_tienda=dict(tienda)
ofertas=dict([["jamón",6],["sal",1.5]])
# -> nuevo diccionario {'jamón': 6, 'sal': 1.5}
lenguajes = ['Java', 'Python', 'JavaScript']
versiones = [14, 3, 6]
lenguajes=dict(zip(lenguajes,versiones))
# -> {'Java': 14, 'Python': 3, 'JavaScript': 6}
```

FUNCIONES UTILES

- ❑ min y max buscan sobre las claves
- ❑ Como objeto que es los diccionarios son referencias.
- ❑ dict permite clonar y crear un diccionario a través de colecciones compatibles
- ❑ zip permite crear un diccionario a partir de listas

MODIFICACION

- ❑ Disponemos de la función del para borrar una clave concreta
- ❑ clear borra todo y update añade un diccionario al actual

```
del tienda["sandia"]  
tienda.update(ofertas) #Añade al diccionario  
tienda.clear()
```



DICCIONARIOS A PARTIR DE OTROS

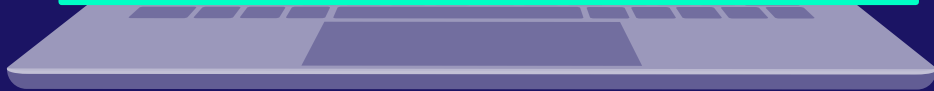
- ❑ Mediante el operador `**` podemos juntar diccionarios.
- ❑ Tengamos en cuenta que diccionarios resultante es uno nuevo no vinculado a ninguno de los anteriores
- ❑ Útil para programación funcional y comprensión de listas que veremos en el siguiente tema

```
#interesante sobre todo en programacion funcional
datos={"h":0,"g":9}
copia_datos={**datos} #es una copia {'h': 0, 'g': 9}

mezcla={**datos,**{'r':90}}
# incorporamos datos a un diccionario y
# resulta uno nuevo {'h': 0, 'g': 9, 'r': 90}

modificacion={**datos,**{'g':90}}
#creamos un diccionario nuevo igual
#pero con la clave modificada

otro_dicc={'r':90}
mezcla={**datos,**otro_dicc}
# resulta uno nuevo {'h': 0, 'g': 9, 'r': 90}
```



COLECCIONES DE UN DICCIONARIO

- ❑ Podemos obtener colecciones de todos los componentes de un diccionario para poder iterar.
- ❑ Estas funciones devuelven colecciones que no están concretadas a lista o tupla por lo tanto hay que transformarlas previamente

```
tuple(tienda.keys())
('patatas', 'cocacola', 'lechuga', 'melon', 'sandia')
tuple(tienda.values())
(20.5, 2.5, 5, 4.3, 5)
list(tienda.items())
[('patatas', 20.5), ('cocacola', 2.5), ('lechuga', 5), ('melon', 4.3), ('sandia', 5)]
#Pares claves-valor como tuplas
```

RECORRIDO

- ❑ Tenemos distintas combinaciones en función de nuestras necesidades.
- ❑ Los pares se guardan mejor como tuplas que es más eficiente y es de un solo uso.
- ❑ No tiene sentido como listas que además si se modifica que sentido tendría el diccionario es otra variable distinta

```
for k in tienda.keys():  
    print(k)  
  
for v in tienda.values():  
    print(v)  
  
for par in tienda.values():  
    print(f"{par[0]} vale {par[1]} euros")  
  
for (k,v) in tienda.items():  
    print(f"{k} vale {v} euros")
```




EJEMPLO DE APLICACION

```
texto="split() es un método incorporado en Python que separa las palabras "\
      "dentro de una cadena usando un separador específico y devuelve un "\
      "array de cadenas Este método acepta como máximo dos parámetros como argumento"
contadores={}
for palabra in texto.split(" "):
    contadores[palabra]=contadores.get(palabra,0)+1

for (palabra,repeticiones) in contadores.items():
    print(f"{palabra} repetida {repeticiones} veces")

mas_repetida=tuple(contadores.items())[0]
for (palabra,repeticiones) in contadores.items():
    if(repeticiones>mas_repetida[1]):
        mas_repetida=(palabra,repeticiones)
print(mas_repetida[0])
```

```
#Solucionado en una linea
print(max(contadores,key=contadores.get))

temperaturas={}
#... meter datos de temp
min(temperaturas,key=temperaturas.get)

ordenado_palabras=sorted(contadores)
ordenado_repeticiones=sorted(contadores,key=contadores.get)
```

MÁS COMPACTO

- ❑ Las funciones max min y sorted admiten un parámetro que es key.
- ❑ Su valor es una función que se aplica a cada par del diccionario.
- ❑ El resultado de dicha función es lo que se usará como criterio para conseguir el objetivo

ESTRUCTURAS COMPLEJAS

❑ Lista de diccionarios

```
frutas=[{  
    "nombre":"Manzana",  
    "precio":2.4,  
    "categoria":"A"  
},  
{  
    "nombre":"Peras",  
    "precio":1.3,  
    "categoria":"B"  
},  
{  
    "nombre":"Naranjas",  
    "precio":4.4,  
    "categoria":"C"  
}]
```

```
fruta_buscada=input("Nombre de la fruta que busca:")  
buscado=None  
for f in frutas:  
    if f.get("nombre")==fruta_buscada:  
        buscado=f  
        break;  
if buscado!=None:  
    print(buscado["precio"])  
else:  
    print("No hay esa fruta")
```

ESTRUCTURAS COMPLEJAS

❑ Diccionario de diccionarios

```
frutas={"Manzana":{  
    "nombre":"Manzana",  
    "precio":2.4,  
    "categoria":"A"},  
    "Peras":{  
    "nombre":"Peras",  
    "precio":1.3,  
    "categoria":"B"},  
    "Naranjas":{  
    "nombre":"Naranjas",  
    "precio":4.4,  
    "categoria":"C"}  
}
```

```
fruta_buscada=input("Nombre de la fruta que busca:")  
print(frutas.get(fruta_buscada,"No existe ese producto"))
```

EJERCICIOS PROPUESTOS

EJERCICIO 24

Crear un programa que dado un numero entero nos muestre cada uno de sus dígitos escritos con palabras. Comprobar que la entrada es numérica.

EJERCICIO 25

Crear un programa que dado un texto con emojis expresados como string imprima el mismo texto pero con los emojis en forma directa, por ejemplo: I am sad :(-> I am sad 😞. El programa debe al menos admitir 10 emojis distintos.

EJERCICIO 26

Crear un programa que reciba una serie de nombres separados por comas y después una serie de números que representan edades. El programa debe generar un diccionario donde la clave son los nombres y los valores las edades. Primero intentarlo usando bucles y después otra versión usando la función zip

EJERCICIOS PROPUESTOS

EJERCICIO 27

Crear un programa que partiendo del siguiente texto

```
""nif;nombre;email;teléfono;descuento\n01234567L;Luis  
González;luisgonzalez@mail.com;656343576;12.5\n71476342J;Macarena  
Ramírez;macarena@mail.com;692839321;8\n63823376M;Juan José  
Martínez;juanjo@mail.com;664888233;5.2\n98376547F;Carmen  
Sánchez;carmen@mail.com;667677855;15.7""
```

Lo convierta a la siguiente estructura en Python:

```
[  
    {'nombre': 'Luis González', 'email': 'luisgonzalez@mail.com', 'teléfono': '656343576', 'descuento': 12.5},  
    {'nombre': 'Macarena Ramírez', 'email': 'macarena@mail.com', 'teléfono': '692839321', 'descuento': 8.0},  
]
```

Hacer otra versión creando la siguiente estructura:

```
{  
    '01234567L':  
        {'nombre': 'Luis González', 'email': 'luisgonzalez@mail.com', 'teléfono': '656343576', 'descuento': 12.5},  
    '71476342J':  
        {'nombre': 'Macarena Ramírez', 'email': 'macarena@mail.com', 'teléfono': '692839321', 'descuento': 8.0},  
}
```

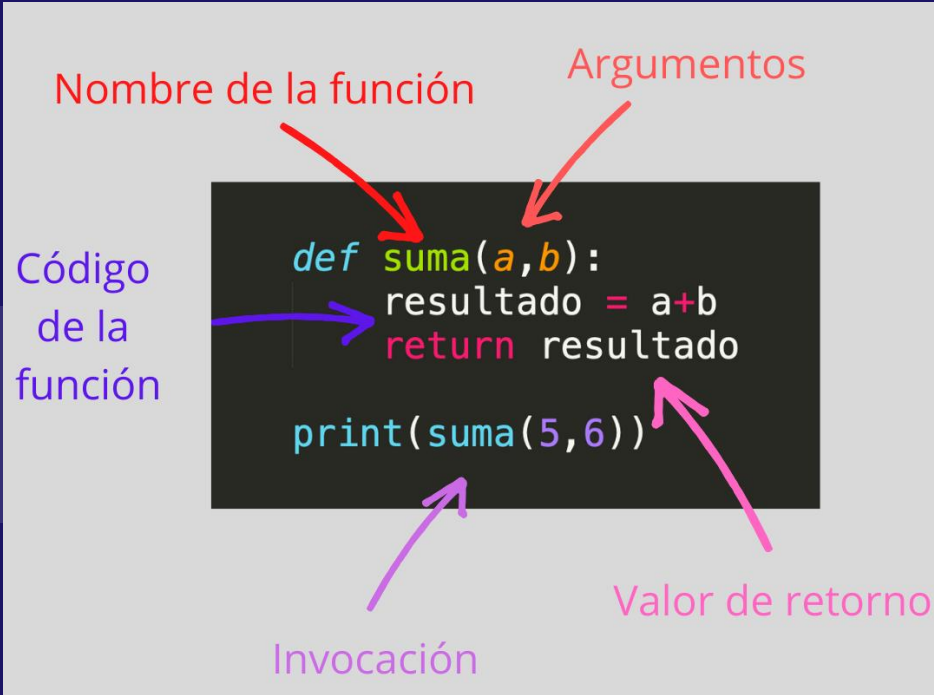
FUNCIONES



DEFINICION DE FUNCIONES

- ❑ Bloque de código reutilizable que realiza una sola tarea específica
- ❑ Hacen que el código sea reutilizable, conciso, legible, mantenible y testeable
- ❑ Tiene asociado un nombre, de manera que cada vez que se quiera ejecutar el bloque de código basta con invocar el nombre de la función

DEFINICION DE FUNCIONES



EJEMPLOS DE FUNCIONES

- ❑ Se pueden devolver colecciones.
- ❑ Puede ser utilidad en ciertos casos y hay módulos de la biblioteca estándar de Python que lo hacen

```
def medidas_cuadrado(lado):  
    area=lado**2  
    perimetro=lado*4  
  
    return (area,perimetro)  
  
def medidas_cuadrado(lado):  
    area=lado**2  
    perimetro=lado*4  
  
    return {"area":area,"perimetro":perimetro}
```

EJEMPLOS DE FUNCIONES

- ❑ El return no es obligatorio aunque suele ponerse por defecto. Yo no lo hago
- ❑ Si queremos una función sin código porque estamos esquematizando la instrucción pass no hace nada

```
def area_triangulo(base, altura):  
    return base * altura / 2  
  
def bienvenida():  
    print('¡Bienvenido a Python!')  
  
def pordefinir():  
    pass
```

LLAMADAS Y PASO DE PARAMETROS

- ❑ Python pasar por parámetros por nombre en lugar de por posición, además de establecer parámetros por defecto

```
def bienvenida(nombre, lenguaje = 'Python'):
    print('¡Bienvenido a', lenguaje, nombre + '!')

bienvenida('Daniel') # -> Bienvenido a Python Daniel!
bienvenida('Daniel', 'Java') # -> Bienvenido a Java Daniel!

#Llamadas por nombre y no por posicion
bienvenida(lenguaje='C++', nombre='Marta') # -> Bienvenido a C++ Marta!
```

PARAMETROS REST

- ❑ Dispone de mecanismos para que el numero de parámetro se adapte a las necesidades usando tuplas o diccionarios

```
#Si quiero la media de mas de 2 numeros  
#y no quiero una lista o tupla  
def calcula_media(x, y):  
    resultado = (x + y) / 2  
    return resultado
```

```
#datos como una tupla  
def calcula_media(*datos):  
    total = 0  
    for i in datos:  
        total += i  
    resultado = total / len(datos)  
    return resultado  
calcula_media(3,5,10)  
calcula_media(1,2,3,4,5)
```

PARAMETROS REST

- ❑ También se pueden introducir datos y que lleguen como un diccionario

```
#Datos como un diccionario
def mayor_precio(**datos):
    producto_mayor=max(datos,key=datos.get)
    return datos[producto_mayor]

mayor_precio(pizza=3.5,cocacola=1.4,patatas=2.3)
```

AMBITO DE LAS VARIABLES

- ❑ Desde una función se accede a la variable más cercana en el bloque
- ❑ Si la variable es un objeto o una colección como son por referencia cualquier cambio afecta a las variables

```
lenguaje = 'Java'  
def bienvenida():  
    lenguaje = 'Python'  
    print(lenguaje)  
print(lenguaje)
```

```
primer_curso = ['Matemáticas', 'Física']  
def añade_asignatura(curso, asignatura):  
    curso.append(asignatura)  
  
añade_asignatura(primer_curso, 'Química')  
print(primer_curso) #-> ['Matemáticas', 'Física', 'Química']
```

FUNCIONES COMO TIPO DE DATO

- ❑ Las funciones son como cualquier tipo de dato ya que Python permite la programación funcional
- ❑ Esto nos permite pasarle parámetros funciones que adaptan la acción a realizar.

```
frutas=[{
    "nombre": "Manzana",
    "precio": 2.4,
    "categoria": "A"
}, {
    "nombre": "Peras",
    "precio": 1.3,
    "categoria": "B"
}, {
    "nombre": "Naranjas",
    "precio": 4.4,
    "categoria": "C"
}]
```

```
def criterio(producto):
    return producto["precio"]

max(frutas, key=criterio)
```

FUNCIONES ANONIMAS

- ❑ Para no tener que definir una función de una sola línea, las funciones lambda son funciones sin nombre que permiten incrustarse en una llamada a otra función

```
max(frutas,key=lambda f: f["precio"])
min(frutas,key=lambda f: f["precio"])
sorted(frutas,key=lambda f: f["precio"])
sorted(frutas,key=lambda f: f["precio"],reverse=True)
```


MODULO 0 EJECTABLE

- ❑ Existe una variable global llamada `__name__` que indica si el fichero es ejecutado o importado
- ❑ Puede que un fichero solo incluya funciones.
- ❑ Puede que incluya funciones y código.
- ❑ Si el fichero se ejecuta como principal `__name__` vale `"__main__"`
- ❑ Esto lo podemos utilizar para que solo se interprete el código necesario para que el programa funciones

MAIN

- ❑ En el primer caso se lee todo el código, en el segundo caso se lee `area_triangulo(2,5)`

```
def area_triangulo(base, altura):  
    return base * altura / 2  
  
def bienvenida(nombre, lenguaje = 'Python'):  
    print('¡Bienvenido a', lenguaje, nombre + '!')  
  
print(area_triangulo(5,7))
```

```
def area_triangulo(base, altura):  
    return base * altura / 2  
  
def bienvenida(nombre, lenguaje = 'Python'):  
    print('¡Bienvenido a', lenguaje, nombre + '!')  
  
if __name__=="__main__":  
    print(area_triangulo(5,7))
```

MAIN

- ❑ También lo que se suele hacer es crearse una función main que contenga el código a ejecutar de manera que es más legible dicha parte

```
def main():
    datos=input("Datos separados por espacios:")
    datos=datos.split(" ")
    total = 0
    for i in datos:
        total += i
    resultado = total / len(datos)
    print("La media es",resultado)

#...
if __name__=="__main__":
    main()
```

EJERCICIOS PROPUESTOS

EJERCICIO 28

Escribir una función que calcule el máximo común divisor de dos números y otra que calcule el mínimo común múltiplo.

EJERCICIO 29

Escribir una función que dado un entero devuelva el mismo número, pero dado la vuelta (no vale devolver un string).

EJERCICIO 30

Escribir una función que reciba una cadena de caracteres y devuelva un diccionario con cada palabra que contiene y su frecuencia. Escribir otra función que reciba el diccionario generado con la función anterior y devuelva una tupla con la palabra más repetida y su frecuencia.

EJERCICIO 31

Escribir un programa que dada la estructura creada en el ejercicio 27 nos muestre el cliente con mayor descuento, una por cada versión de la estructura. Usa funciones de lista y lambda

Bibliografía y enlaces

APUNTES MANUALES Y TUTORIALES:

- Autor original: Guido van Rossum, Fred L. Drake, Jr "El tutorial de Python". Python Software Foundation 2018
- Documentación de usuario de Odoo
<https://www.odoo.com/documentation/user/15.0/es/>
- Documentación del desarrollador de Odoo
<https://www.odoo.com/documentation/15.0/es/>

VIDEO-TUTORIALES:

- Python For Everybody
<https://www.youtube.com/watch?v=o0XbHvKxw7Y>
- TechWorld with Nana
<https://www.youtube.com/watch?v=t8pPdKYpowI>
- Freecodecamp
<https://www.youtube.com/watch?v=DLikpfc64cA>
- Píldoras informáticas
<https://www.youtube.com/watch?v=G2FCfQj-9ig&list=PLU8oAlHdN5BlvPxziopYZRd55pdqFwkeS>

FIN DEL TEMA 3!

Cualquier duda enviar un correo a
daniel.lopez@escuelaartegranada.com

CREDITS: This presentation template was created by
Slidesgo, including icons by Flaticon, and
infographics & images by Freepik.