

- Cadenas(Strings) en Python
- Segmentos de Cadenas
- Métodos de formato:
 - **title**
 - **capitalize**
 - **lower**
 - **upper**
 - **Otros:** swapcase, center, ljust, rjust, zfill
- Métodos de Búsqueda
 - **count**
 - **find**
- Métodos de Validación
 - **isspace**
 - **isalpha**
 - **Otros:** isdigit, islower, isupper, istitle, isalum
- Métodos de Sustitución
 - **replace**
 - **Otros:** strip,rstrip, lstrip
- Métodos de unión y división
 - **split**
 - **join**

1) Listas

2) Estructuras repetitivas

- **Para**
- **Mientras**
- **Repetir**

3) Estructuras repetitivas en Python

- **For in**
- **While**

TEMAS DE CLASE

TEORÍA 2



Strings en Python

- Las cadenas están compuestas de piezas más pequeñas—los caracteres.
- Los tipos que comprenden piezas más pequeñas se denominan **tipos de datos compuestos**.
- Podemos tratar un tipo compuesto como unidad o podemos acceder a sus partes.
- El operador corchete selecciona un carácter de una cadena.
- La expresión en corchetes se denomina **índice**.

EJEMPLO:

```
>>> fruta = "manzana"  
>>> letra = fruta[1]  
>>> print (letra)
```



Segmentos de cadenas

- Una porción de una cadena de caracteres se denomina **segmento**.
- El operador [n:m] retorna la parte de la cadena que va desde el carácter n hasta el m, incluyendo el primero y excluyendo el último.
- Si se omite el primer índice (antes de los puntos seguidos), el segmento comienza en el inicio de la cadena.
- Si se omite el segundo índice, el segmento va hasta el final.

Ejemplos:

```
>>> s = "Pedro, Pablo, y Maria"
```

```
>>> print s[0:5]
```

```
Pedro
```

```
>>> print s[7:12]
```

```
Pablo
```

```
>>> print s[16:21]
```

```
Maria
```



MÉTODOS DE FORMATO

Método: title()

- Convierte una cadena a formato título

EJEMPLO:

```
>>> cadena = "hola mundo"
```

```
>>> print(cadena.title())
```

Hola Mundo

Método: capitalize()

Convierte a mayúscula la primera letra de una cadena

EJEMPLO:

```
>>> cadena = "bienvenido a mi aplicación"
```

```
>>> print(cadena.capitalize())
```

Bienvenido a mi aplicación



MÉTODOS DE FORMATO

Método: lower()

- Convierte una cadena a minúsculas

EJEMPLO:

```
>>> cadena = "Hola Mundo"
```

```
>>> print(cadena.lower())
```

hola mundo

Método: upper()

- Convierte una cadena a mayúsculas

EJEMPLO:

```
>>> cadena = "Hola Mundo"
```

```
>>> print(cadena.upper())
```

HOLA MUNDO



MÉTODOS DE BÚSQUEDA

Método: count("subcadena" [, posicion_inicio, posicion_fin])

- Retorna un entero representando la cantidad de apariciones de “subcadena” dentro de cadena.

- EJEMPLO:

```
>>> cadena = “bienvenido a mi aplicación”
```

```
>>> print(cadena.count(“a”))
```

3

Método: find("subcadena" [, posicion_inicio, posicion_fin])

- Retorna un entero representando la posición donde inicia la “subcadena” dentro de cadena. Si no la encuentra, retorna -1.

- EJEMPLO:

```
>>>cadena = “bienvenido a mi aplicación”
```

```
>>> print cadena.find("mi")
```

13



MÉTODOS DE VALIDACIÓN

Método: isalpha()

- Permite saber si una cadena es alfabética
- Retorna: True o False
- EJEMPLO:

```
>>> cadena = "programa 5"
```

```
>>> print(cadena.isalpha())
```

False

```
>>> cadena = "programa"
```

```
>>> print(cadena.isalpha())
```

True

Método: isspace()

- Permite saber si una cadena contiene solo espacios en blanco
- Retorna: True o False
- EJEMPLO:

```
>>> cadena = «programa 5"
```

```
>>> print(cadena.isspace())
```

False

```
>>> cadena = " "
```

```
>>> print cadena.isspace()
```

True



MÉTODOS DE SUSTITUCIÓN

Método: `replace("subcadena a buscar", "subcadena por la cual reemplazar")`

- Permite reemplazar texto en una cadena.
- Devuelve la cadena reemplazada
- EJEMPLO:

```
>>> buscar = "nombre apellido"
```

```
>>> reemplazar_por = "Juan Pérez"
```

```
>>> print ("Estimado Sr. nombre apellido:".replace(buscar, reemplazar_por))
```

Estimado Sr. Juan Pérez:



MÉTODO DE DIVISIÓN

Método: `split("separador")`

- Divide una cadena en varias partes utilizando un separador
- Devuelve una lista con todos elementos encontrados al dividir la cadena por un separador.
- EJEMPLO:

```
>>> frase = "python, guia, curso, tutorial".split(", ")
```

```
>>> print (frase)
```

```
['python', 'guia', 'curso', 'tutorial']
```



LISTAS

- Una LISTA es una **variable** que **permite almacenar varios datos**.
- Los elementos de una lista no tienen que tener el mismo tipo.
- Permite modificar los datos una vez creados

Ejemplo:

```
mi_lista = ['cadena de texto', 15, 2.8, 'otro dato', 25]
```

- A las listas se ***accede*** por su número de ***índice***:

```
print(mi_lista[1]) # Salida: 15
```

```
print(mi_lista[1:4]) # Devuelve: [15, 2.8, 'otro dato']
```

```
print(mi_lista[-2]) # Salida: otro dato
```

- Permite modificar los datos una vez creados

```
mi_lista[2] = 3.8 # el tercer elemento ahora es 3.8
```

- Las listas permiten agregar nuevos valores con el método ***append***:

```
mi_lista.append('Nuevo Dato')
```



ESTRUCTURAS REPETITIVAS

- Permiten ejecutar un mismo fragmento de código un número determinado de veces. Se les denomina bucles.
- En algunos algoritmos podemos establecer a priori que el bucle se repetirá un número de veces. Es decir, el número de repeticiones no dependerá de las proposiciones dentro del ciclo, a esta estructura se le conoce como PARA.
- En algunos algoritmos no podemos establecer a priori cuantas veces se repetirá el bucle, sino que dependerá de las proposiciones dentro del mismo. En este grupo de estructuras se encuentra MIENTRAS y REPETIR.



Estructura repetitiva: Mientras

- La condición del bucle se **evalúa al principio, antes de entrar en él**. *Si la condición es verdadera, comenzamos a ejecutar las acciones del bucle y después de la última volvemos a preguntar por la condición.* Si la condición es **falsa**, se sale de la sentencia **Mientras** y continúa la ejecución con la siguiente sentencia (afuera del Mientras). Evalúa la condición, resultando en False (falso) o True (cierto).

Pseudocódigo:

Mientras <condición> **hacer**

<operaciones>

.....

.....

Fin Mientras



Estructura repetitiva: Para

- Este tipo de bucles se utiliza cuando se sabe ya antes de ejecutar el bucle **el número exacto de veces que hay que ejecutarlo.**

Pseudocódigo:

Para <var índice> =<vi> hasta <vf>

<operaciones>

.....

.....

Fin Para



Estructura repetitiva: Repetir

- El bucle repite mientras la condición sea falsa. **La condición se evalúa siempre al final del bucle**, si es falsa volvemos a ejecutar las acciones, si es verdad se sale del bucle.
- Cuando **un bucle se tenga que ejecutar como mínimo una vez**, podremos usar una estructura repetir.

Repetir

<Operaciones>

...

...

hasta que <condición



- El bucle **While** (mientras) ejecuta un fragmento de código mientras se cumpla una condición.

Ejemplo 1:

```
i = 1
while i < 10:
    print(i)
    i = i + 1 #contador
```

Ejemplo 2:

```
edad = 0
while edad < 18:
    print ("Felicidades, tienes " + str(edad))
    edad = edad + 1
```

WHILE

Estructuras repetitivas en
Python



- En Python **For** se utiliza como una forma de iterar sobre una secuencia.
- Significa que necesita de una lista para iterar .
- Itera tantos elementos tenga la lista.

Ejemplo 1:

```
lista = [1,3,5,2,4,7,8,6,9]
#este for imprime por pantalla los valores de la lista
for elemento in lista:
    #En cada iteración ELEMENTO toma un valor de LISTA
    print(elemento)
```

Ejemplo 2:

```
mi_lista = ['Juan', 'Antonio', 'Pedro', 'Herminio']
for nombre in mi_lista:
    print (nombre)
```

FOR IN

Estructuras repetitivas en Python



FUNCIÓN RANGE

- RANGE es una función que crea lista de acuerdo a sus parámetros

#si tiene un solo parámetro: de cero hasta menor que el numero

range(10) #Crea una lista desde 0 hasta 9: [0,1,2,3,4,5,6,7,8,9]

#range(inicio, fin)

range(1, 10) #Crea una lista desde 1 hasta 9: [1,2,3,4,5,6,7,8,9]

#range(inicio, fin, incremento/decremento)

range(1, 10, 2)

#Crea una lista desde 1 a 9 de 2 en 2: [1,3,5,7,9]

#El último elemento no se toma en cuenta. Si es 10, se cuenta hasta 9



FOR Y RANGE

Se usa en situaciones en el que queremos que un **For** itere cierta cantidad de veces.

Ejemplo:

```
for i in range(10):  
    print(i)
```

Funcionamiento

- Range crea una LISTA de acuerdo con sus parámetros
- En cada iteración *i* toma un valor de la LISTA



BREAK

- Sentencia que permite Salir de un Bucle

Ejemplos:

```
x = 5
```

```
while True:
```

```
    x -= 1
```

```
    print(x)
```

```
    if x == 0:
```

```
        break
```

```
cadena = 'Python'
```

```
for letra in cadena:
```

```
    if letra == 'h':
```

```
        print("Se encontró la h")
```

```
        break
```

```
    print(letra)
```

CONTINUE

- Permite detener la iteración actual y continuar con la siguiente.

Ejemplo:

```
cadena = 'Python'
```

```
for letra in cadena:
```

```
    if letra == 'P':
```

```
        continue
```

```
    print(letra)
```



Recorrido de Cadenas

- Muchos cálculos implican procesar una cadena **carácter** por carácter.
- Se puede realizar con la sentencia **while** o **for**

```
fruta="mandarina"  
i = 0  
while i < len(fruta):  
    letra = fruta[i]  
    print(letra)  
    i +=1
```

```
fruta="mandarina"  
for caracter in fruta:  
    print(caracter)
```

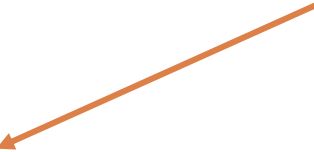


FUNCIÓN RANDINT

La función **randint()** devuelve un número entero incluido entre los valores indicados. Los valores de los límites inferior y superior también pueden aparecer entre los valores devueltos.

En el siguiente ejemplo se realizan cinco "sorteos" y se obtiene, para cada uno de ellos, un regalo de diez posibles.

```
import random
```



El módulo **random** de la librería estándar de Python incluye un conjunto de funciones que permiten obtener de distintos modos números aleatorios.

```
regalos = ['sartén', 'jamón', 'mp4', 'muñeca', 'tv',  
           'patín', 'balón', 'reloj', 'bicicleta', 'anillo']
```

```
for sorteo in range(5):  
    regalo = regalos[random.randint(0, 9)]  
    print('Sorteo', sorteo + 1, ':', regalo)
```

