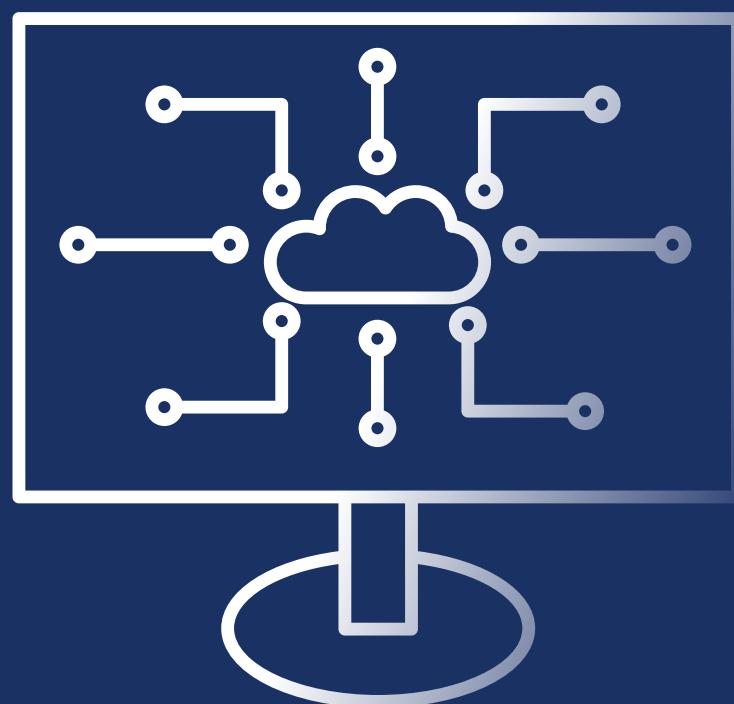


Programación I



Lic. Julia Monasterio
marmonasterio@uade.com.ar



Unidades

-  **Unidad 1:** Repaso de matrices, estructura, operaciones y uso en funciones
-  **Unidad 2:** Implementación y gestión en proyectos en Git
-  **Unidad 3:** Listas avanzadas, cadena de caracteres y expresiones regulares.
-  **Unidad 4:** Manipulación avanzada de diccionarios, tuplas y conjuntos.
-  **Unidad 5:** Excepciones y pruebas unitarias
-  **Unidad 6:** Archivos
-  **Unidad 7:** Recursividad

Clase N°5

TEMAS

- Continuación de Cadena de Caracteres
 - Métodos interrogativos de cadenas de caracteres
 - Otros métodos
 - Métodos de formateo
- Practica de cadena de caracteres
- Funciones Lambda
- Expresiones Regulares
-

Listas y cadenas

Similitudes entre listas y cadenas

- Ambas son secuencias de elementos son iterables, es decir pueden ser recorridas por un **ciclo for**
- Secuencia significa que son iterables que mantienen un determinado orden sus elementos
- Se puede acceder a cada elemento a través de un subíndice (base 0)



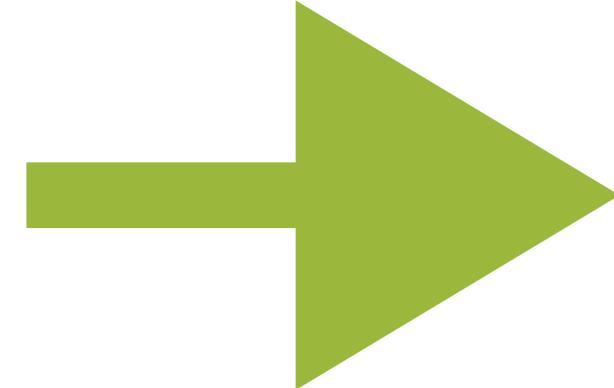
```
cad="Lunes"  
print(cad[3])
```

u

Similitudes entre listas y cadenas

- Pueden manipularse mediante rebanadas

```
cad="Lunes"  
subcadena=cad[1:3]  
print(subcadena)
```

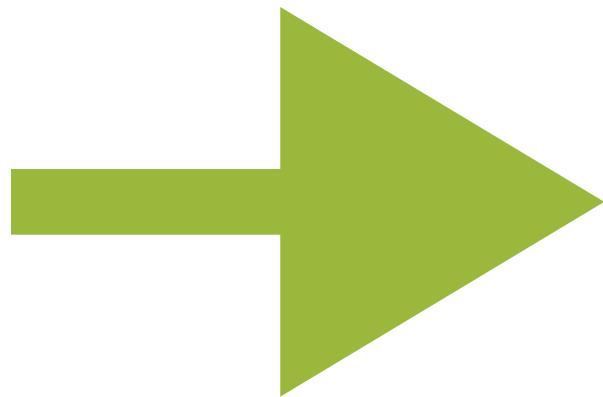


```
un
```

Similitudes entre listas y cadenas

- Pueden concatenarse con el operador +

```
nombre=input("Ingrese su nombre\n")
saludo="Su nombre es "+nombre
print(saludo)
```

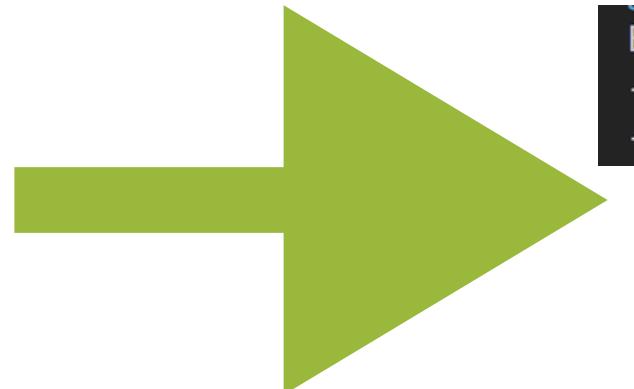


```
Ingrese su nombre
Julia
Su nombre es Julia
```

Similitudes entre listas y cadenas

- Pueden replicarse mediante el operador *

```
conector="Es"  
  
print(conector*5)  
  
separador=" - "*80  
  
print(separador)
```



```
EsEsEsEsEs  
-----  
-----
```

Similitudes entre listas y cadenas

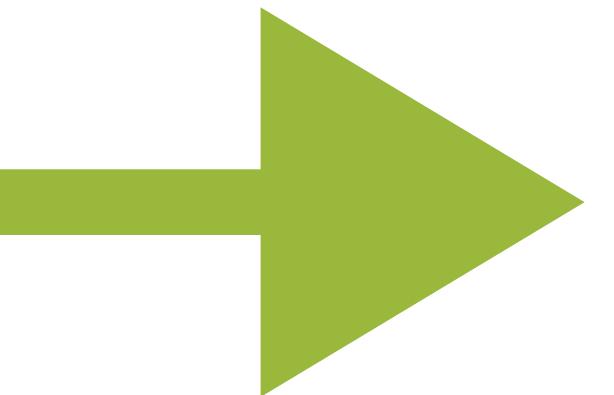
- Comparten funciones y métodos: **len, in, not in, max, min, index, count**

```
cadena="Jueves"

print(len(cadena))

if "ev" in cadena:
    print("Se encuentra")

if "pe" not in cadena:
    print("No se encuentra")
```

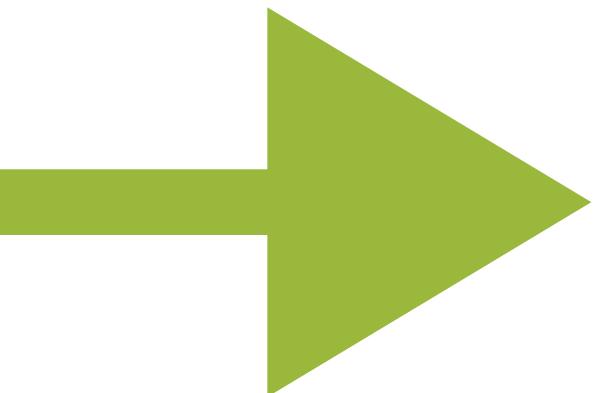


```
6
Se encuentra
No se encuentra
```

Similitudes entre listas y cadenas

- Comparten funciones y métodos: **len, in, not in, max, min, index, count**

```
cadena="Jueves"  
  
print(max(cadena))  
print(min(cadena))  
print(cadena.index("es"))  
print(cadena.count("es"))
```



```
V  
J  
4  
1
```

Diferencias entre cadenas y listas

- Las listas son mutables, pero las cadenas son **inmutables**
- Esto significa que una vez creadas **no pueden ser modificadas**
- Las funciones y métodos para manipularlas devuelven una nueva cadena, sin alterar la original

Diferencias entre cadenas y listas

```
cadena="Hola Pepe"
```

```
cadena[5]="X"
```

```
py , line 5, in <module>
    cadena[5]="X"
~~~~~^~~~
```

```
TypeError: 'str' object does not support item assignment
```

Diferencias entre cadenas y listas

- Debido a que son inmutables, las cadenas carecen del método **reverse** de las listas.
- Para invertir una cadena se puede utilizar la técnica de las rebanadas.

```
cadena="Jueves"  
  
invertida=cadena[::-1]  
  
print(invertida)
```

Práctica en clase

Dibujar el borde de un cuadrado con letras X, ingresando por teclado la longitud de cada lado



Resolución ejercicio

```
bandera=True
while bandera:
    lado=int(input("Ingrese un valor de lado. Debe ser mayor que tres\n"))
    if lado<3:
        print("Lontigud incorrecta. Intente nuevamente\n")
    else:
        bandera=False

    print("X"*lado) #Borde superior

    relleno="X"+" "* (lado-2)+ "X"

    for i in range(lado-2):#Se le resta dos porque no vamos a imprimir el borde
superior y el inferior
        print(relleno)

    print("X"*lado)
```

Comparación

- Las cadenas de caracteres pueden ser comparadas entre si igual que cualquier otra variable
- Se utiliza comparación alfabética, es decir que las palabras se clasifican como en un diccionario

Comparación

Funciones **chr** y **ord** se utilizan para convertir entre valores numéricos y caracteres ASCII.

- La función **chr()** toma un numero entero como argumento y devuelve el carácter correspondiente a ese numero
- La función **ord()** toma un caracter como argumento y devuelve el valor numerico correspondiente.

Comparación

```
for i in range(65,75):  
    print(i,chr(i))
```

```
65 A  
66 B  
67 C  
68 D  
69 E  
70 F  
71 G  
72 H  
73 I  
74 J
```

```
print(ord('a'))
```

97

Conversion

- Un numero no puede ser concatenado a una cadena de forma directa

```
precio=75
```

```
mensaje="El precio total es "+precio
```

- Antes es necesario convertirlo a cadena, lo que se logra con la funcion **str()**

```
precio=75
```

```
mensaje="El precio total es "+str(precio)
```

```
print(mensaje)
```

3, clase 3, practica linea

El precio total es 75

Conversion

- Así como existe la función **str()**, para convertir números en cadenas, existen las funciones **int()** y **float()** que permiten efectuar la conversión en sentido contrario.
- **Si la cadena no contiene un numero valido se producirá un error**

Práctica en clase

Reemplazar las vocales con tilde en una cadena de caracteres por sus equivalentes sin tilde



Resolución ejercicio

```
conTilde="ÁÉÍÓÚáéíóú"  
sinTilde="AEIOUaeiou"  
  
frase=input("Ingrese una frase\n")  
cadenaConReemplazo=""  
  
for caracter in frase:  
    if caracter in conTilde:  
        posicion=conTilde.index(caracter)  
        cadenaConReemplazo=cadenaConReemplazo+sinTilde[posicion]  
    else:  
        cadenaConReemplazo=cadenaConReemplazo+caracter  
  
print(cadenaConReemplazo)
```

Ingrese una frase
pero vos estás segúro?
pero vos estas seguro?

Métodos para cadena

Métodos interrogativos

Métodos para cadenas

- **Métodos interrogativos:** son métodos donde los nombres de los mismos comienzan con **is**
- Se trata de métodos interrogativos, que no llevan parámetros y permiten obtener información acerca de una cadena.
- **Todos los métodos que comienzan con is devuelven True o False**

Métodos para cadenas

- **Metodo `<str>.isalpha()`:** devuelve `True` si **todos** los caracteres de `<str>` son alfabéticos (letras), o `False` en caso contrario. Reconoce caracteres regionales

```
cadena1="Pepe1"  
  
print(cadena1.isalpha())  
  
cadena2="pepe"  
print(cadena2.isalpha())  
  
cadena3="Hola como estas?"  
print(cadena3.isalpha())
```

False
True
False

Métodos para cadenas

- **Metodo `<str>.isdigit()`:** devuelve True si **todos** los caracteres de `<str>` son dígitos numéricos, o False en caso contrario.

```
cadena1="1234"  
cadena2="3.14"
```

```
print(cadena1.isdigit(),cadena2.isdigit())
```

```
: True False
```

Métodos para cadenas

- **Metodo <str>.isalnum():** devuelve True si **todos** los caracteres de <str> son dígitos numéricos o alfabeticos, o False en caso contrario.

```
cadena1="-1234"  
cadena2="AF512SP"  
  
print(cadena1.isalnum(),cadena2.isalnum())
```

False True

Métodos para cadenas

- **Método `<str>.isupper()`:** devuelve True si **todos** los caracteres alfabéticos de `<str>` están en mayúscula. Ignora los caracteres no alfabéticos.

```
cadena="HOLA COMO ANDAS"  
  
if cadena.isupper():  
    print("Esta en mayuscula ")
```

Métodos para cadenas

- **Método `<str>.islower()`:** devuelve True si **todos** los caracteres alfabéticos de `<str>` están en minúscula. Ignora los caracteres no alfabéticos.

```
cadena="prueba de minuscula"

if cadena.islower():
    print("Esta en minuscula ")
```

Práctica en clase

Escribir una función para separar una palabra de los signos de puntuación que pueda tener, devolviendo tres strings: prefijo, palabra, sufijo

Ejemplo:

cad=""(ingeniero)"

limpiarCadena(cad) -> '(',')ingeniero',''



Resolución ejercicio - Función

```
def limpiarPalabra(palabra):
    indiceInicial=0

    while indiceInicial<len(palabra) and not palabra[indiceInicial].isalpha():
        indiceInicial+=1
    inicio=palabra[:indiceInicial]

    indiceFinal=len(palabra)-1

    while indiceFinal>indiceInicial and not palabra[indiceFinal].isalpha():
        indiceFinal-=1

    final=palabra[indiceFinal+1:]
    palabra=palabra[indiceInicial:indiceFinal+1]

    return inicio,final,palabra
```

Resolución ejercicio - Principal

```
#Programa Principal  
  
palabraPrueba=".Pepe."  
print(limpiarPalabra(palabraPrueba))
```

Otros métodos

Métodos para cadenas

- **Método <str>.upper():** devuelve <str> convertida a mayúsculas.
Los caracteres alfabéticos no resultan modificados
- Recuerden que los strings son **inmutables** por ende el método devuelve una copia modificada

```
cadena= "cambiame a mayus :)"  
print(cadena.upper())
```

CAMBIAME A MAYUS :)

Métodos para cadenas

- **Método `<str>.lower()`:** devuelve `<str>` convertida a minúsculas.
Los caracteres alfabéticos no resultan modificados

```
cadena= "ESTO ES UNA ORDEN!!! 0:"
```

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

```
estos es una orden!!! 0:
```

Métodos para cadenas

- **Método `<str>.swapcase()`:** devuelve `<str>` intercambiando mayúsculas por minúsculas y viceversa

```
cadena="h0LA c0MO aNDAS"  
  
cadena2=cadena.swapcase()  
  
print(cadena2)
```

Hola Como Andas

Métodos para cadenas

- **Método <str>.replace(cadVieja,cadNueva):** devuelve <str> cambiando la primera ocurrencia del carácter especificado en el primer parámetro por el especificado en el segundo

```
cadena="hola mundo"  
  
cadena2=cadena.replace("hola","chau")  
  
print(cadena2)
```

chau mundo

Métodos para cadenas

- **Método `<str>.capitalize()`:** devuelve `<str>` convirtiendo a mayúscula el primer carácter de la cadena, y todo lo demás a minúscula

```
cadena= "esta es una oracion"  
print(cadena.capitalize())
```

Esta es una oracion

- **y esto??**

```
cadena= ".esta es una oracion"  
print(cadena.capitalize())
```



Métodos para cadenas

- **Método `<str>.title()`:** devuelve `<str>` convirtiendo a mayúscula el primer carácter de cada palabra, y todo lo demás a minúscula

```
cadena= "bienvenidos a NUESTRA web"  
print(cadena.title())
```

Bienvenidos A Nuestra Web

Métodos para cadenas

- **Método <str>.split(sep, maxsplit)**: devuelve una lista de subcadenas. Dividiendo a la cadena por el **sep**
- **sep** es el separador que queremos utilizar para dividir. Debe especificarse como una cadena
- **maxsplit** es opcional, e indica la cantidad de veces máxima que se quiere dividir la cadena

```
cadena="somos estudiantes de programacion de UADE"  
  
sublistas=cadena.split(" ")  
  
print(sublistas)
```

```
['somos', 'estudiantes', 'de', 'programacion', 'de', 'UADE']
```

Métodos para cadenas

- **Método `<sep>.join(<iterable>)`:** permite formar una cadena a partir de subcadenas
 - **<sep>** es el separador sobre el que queremos que se unan las cadenas
 - **<iterable>** cualquier iterable que contenga subcadenas, por ejemplo lista

```
lista="Manzanas,Naranjas,Peras,Mandarinas,Platanos"

sublistas=lista.split(",")

unido="-".join(sublistas)

print(unido)
```

Práctica en clase

Ingresar por teclado el nombre de una entidad o institución y generar la sigla correspondiente tomando la inicial de cada una de sus palabras.

Por ejemplo:

Automóvil Club Argentino -> ACA

Banco Central Republica Argentina -> BCRA



Resolución 1 ejercicio

```
cadena=input("Ingrese el nombre de la entidad completo\n")

listaPalabras= cadena.split(" ")

sigla=""

for palabra in listaPalabras:
    sigla=sigla+palabra[0].upper()

print(sigla)
```

Resolución 2 ejercicio

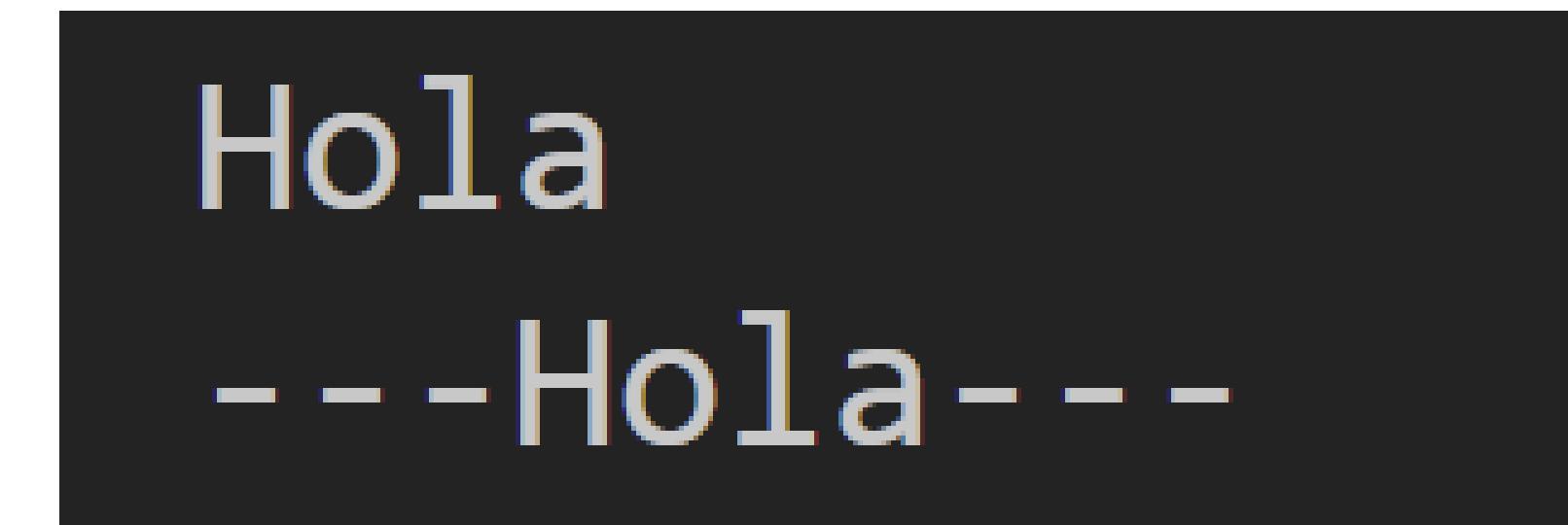
```
sigla="" .join([p[0] for p in cadena.split(" ")]) .upper()  
print(sigla)
```

Métodos para alterar el formato de las cadenas

Métodos para cadenas

- **Método <str>.center(<ancho>[,<relleno>]):** devuelve una cadena centrada en el ancho especificado. El resto de la cadena se rellena con espacios o con el carácter de relleno si esta presente.

```
cadena1="Hola"  
cadena2= cadena1.center(10,'-')  
  
print(cadena1)  
print(cadena2)
```



```
Hola  
- - -Hola-- -
```

Métodos para cadenas

- **Método <str>.ljust(<ancho>[,<relleno>]):** devuelve una cadena alineada a la izquierda en el ancho especificado. El final de la cadena se rellena con espacios o con el carácter de relleno, si esta presente.

```
cadena1="Hola"  
cadena2= cadena1.ljust(10,'-')  
  
print(cadena1)  
print(cadena2)
```



```
Hola  
Hola-----
```

Métodos para cadenas

- **Método <str>.rjust(<ancho>[,<relleno>]):** devuelve una cadena alineada a la derecha en el ancho especificado. El final de la cadena se rellena con espacios o con el carácter de relleno, si esta presente.

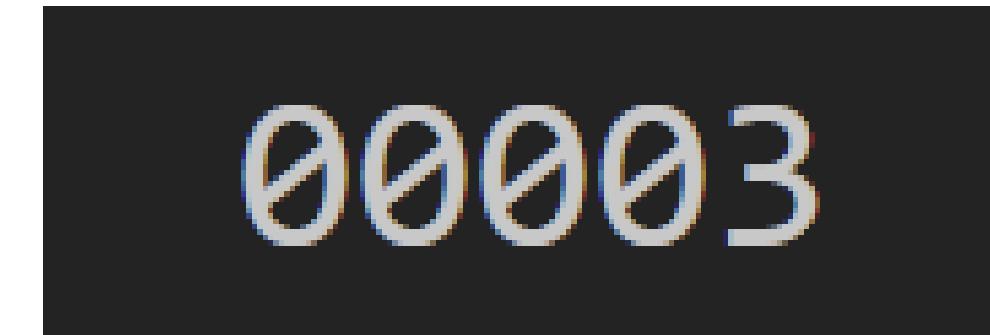
```
practica_recta.ipynb: In [1]: from IPython.display import display, HTML
In [2]: cadena1="Hola"
In [3]: cadena2= cadena1.rjust(10,'-')
In [4]: print(cadena1)
In [5]: print(cadena2)
```

```
Hola
-----Hola
```

Métodos para cadenas

- **Método <str>.zfill(<ancho>):** devuelve una cadena alineada a la derecha en el ancho especificado. El comienzo de la cadena se rellena con ceros.

```
n=3  
cad=str(n).zfill(5)  
print(cad)
```



```
00003
```

Métodos para cadenas

- **Método <str>.lstrip([<cad>]):** elimina todos los caracteres de los especificados en el argumento desde el comienzo de la cadena
- El método continua eliminando caracteres mientras encuentre coincidencias desde la izquierda, y se detiene cuando encuentra el primer carácter que no coincide
- Si <cad> se omite se eliminaran los espacios.

```
cadena1='-->Salida-->'  
cadena2=cadena1.lstrip("->")  
  
print(cadena2)
```

Salida-->

Métodos para cadenas

- **Método `<str>.rstrip([<cad>])`:** elimina todos los caracteres de los especificados en el argumento desde el fin de la cadena
- El método continua eliminando caracteres mientras encuentre coincidencias desde la derecha, y se detiene cuando encuentra el primer carácter que no coincide
- Si `<cad>` se omite se eliminaran los espacios.

Métodos para cadenas

- **Método <str>.strip([<cad>]):** elimina todos los caracteres de los especificados en el argumento desde ambos extremos
- Si <cad> se omite se eliminaran los espacios.

```
lista=[1,2,3,4]

cadena=str(lista)

nueva_cadena=cadena.strip("[]")

print(nueva_cadena)
```

1, 2, 3, 4

Métodos para cadenas

Método <str>.find(<cad> [,<inicio>[,<fin>]]): busca la primera aparición de <cad> dentro de <str>. Devuelve la posición donde se encontró. A diferencia de index, finde no provoca error si no se encuentra sino que devuelve -1

- Pueden indicarse los subíndices donde comenzara y terminara la búsqueda

Métodos para cadenas

Método <str>.rfind(<cad> [,<inicio>[,<fin>]]): similar a find pero busca la ultima aparicion de cad dentro de str.

- Pueden indicarse los subíndices donde comenzara y terminara la búsqueda

Práctica en clase

Desarrollar un programa para imprimir una factura de venta para una empresa.

El objetivo es ver los métodos que vimos, con datos fijos



Resolución ejercicio

```
print("-"*40)
print("Beets S.A".center(40))
print("Carmen Sixta 279".ljust(20),end="")
print("Tel. 4756-0987".rjust(20))
numero=12
print("Factura N° ",str(numero).zfill(8))
articulo="4532 Desarrollo de Aplicacion"
print(articulo.lstrip("0123456789").ljust(30,"."),end="")
precio=1000
print(str(precio).rjust(10,'.'))
print("-"*40)
```

Utilización de cadenas en f-string

Utilizacion de cadenas en f-string

- Pueden utilizar metodos dentro de el format string

```
dia="Lunes"  
print(f'{dia.upper()}')
```

Utilizacion de cadenas en f-string

- Para alinear números y cadenas se puede colocar un numero luego del nombre de la variable, separado por dos puntos. Este numero representara el ancho

```
numero=25  
  
cadena=f"{numero:5}"  
  
print(cadena)
```



25

- Si quiero que se rellene con ceros adelante, coloco el 0 antes que el ancho. **Solo funciona con cero**

Funciones Lambda

Funciones Lambda

- Son funciones pequeñas, anónimas, desecharables y de una sola linea. También se conocen como funciones **anónimas (no es necesario proporcionar un nombre a las funciones lambda)**
- Se pueden usar en cualquier donde se admite una función y viceversa
- Las funciones lambda se escriben en el lugar donde se necesitan

Funciones Lambda

- Se comportan como funciones normales declaradas con la palabra clave **def**. **Pueden contener solo una expresión, por lo que no son las mas adecuadas para funciones con instrucciones de flujo de control**
- A diferencia de las funciones normales, no las definimos con la palabra def, en su lugar se definen en una sola linea que ejecuta una sola expresión
- **Toda función lambda se puede convertir en una función normal, pero no viceversa**

Funciones Lambda - Sintaxis

- Las funciones lambda pueden tener cualquier numero de argumentos pero solo una expresión. Para poder utilizarlas debemos asignarlas a una variable

<var> = lambda <parametros> : <valor de retorno>

La variable ubicada a la izquierda del signo igual pasa a **comportarse como una función**

Funciones Lambda - Ejemplo

- Realizar una función que calcule el cuadrado de un numero

Forma clásica

```
def calcularCuadrado(x):  
    return x**2
```

```
print(calcularCuadrado(2))
```

Función lambda

```
cuadrado=lambda x: x**2
```

```
print(cuadrado(2))
```

Funciones Lambda - Eliminación

- **Como desechamos una función lambda?**

Con el solo hecho de modificar el valor que tiene asignado la variable, la función se pierde.

Funciones Lambda - Ejemplo

- Realizar una función que sume dos números

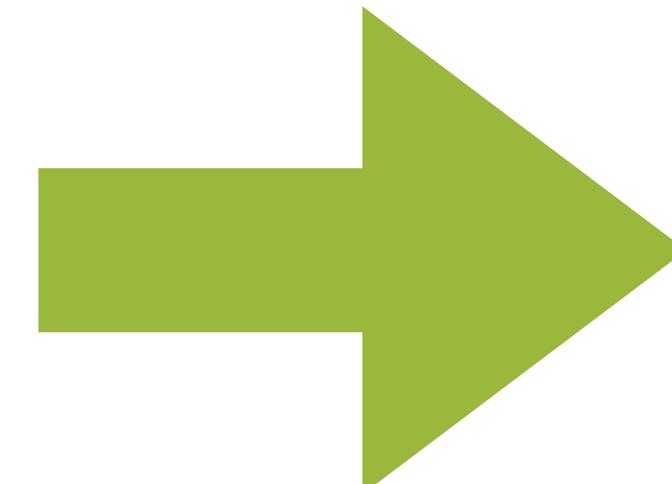
```
suma= lambda x,y: x+y
```

```
print(suma(5,6))
```

Funciones Lambda - Ejemplo

- También pueden usarse parámetros con valores por omisión, o por defecto.
- Es decir si se invoca la función sin ese parámetro se toma el valor por defecto

```
suma= lambda x,y=3: x+y  
print(suma(5))
```



```
8
```

Funciones Lambda - Ejemplo 2

- Realizar una función lambda que compruebe si un numero es impar

```
impar=lambda x: x%2!=0
```

```
print(impar(9))
```

Funciones Lambda - Ejemplo 3

- Invertir una cadena

```
invertir = lambda cadena: cadena[::-1]

print(invertir([1,2,3,4,5]))
```

Funciones Lambda - Invocación inline

- Las funciones lambda se pueden invocar inmediatamente después de su definición utilizando paréntesis para pasar argumentos. Es importante destacar que la variable a la que se le asigna queda con el resultado. Es decir ya no es una función

```
suma = (lambda x,y,z:x+y+z)(9,8,7)  
print(f"El resultado de la suma es {suma}")
```

Funciones Lambda - Otros ejemplos

- Concatenar dos cadenas

```
cadenaResultante=(lambda cadena1,cadena2: cadena1+cadena2)("hola"," como estas?")  
  
print(cadenaResultante)
```

- Elevar un numero a otro

```
potencia=(lambda x,y: x**y)(5,4)  
  
print(potencia)
```

Funciones Lambda - Con if

- Sintaxis:

```
lambda p1,...,pn :{valor condicion verdadera} if condicion else {valor  
condicion falsa}
```

Funciones Lambda - Ejemplo

- Calcular la potencia cuadrada si el numero es mayor a cero

```
cuadradoMayorCero = lambda numero: numero**2 if numero>0 else 0  
  
print(cuadradoMayorCero(-9))  
print(cuadradoMayorCero(2))
```

El cuadrado de -9 es 0
El cuadrado de 2 es 4

Funciones Lambda - Sugerencia

Importante!

Si bien se puede utilizar **if , else** en una expresión lambda, si la lógica es demasiado compleja, puede ser mas legible utilizar una función regular en su lugar.

Función map

Funciones Map

- La función **map** aplica una función cualquiera a todos los elementos de una lista y obtener un conjunto de datos ya transformados
- Su sintaxis es la siguiente:

<lista2> = list(map(<funcion>,<lista1>))

Funcion Map - Ejemplo

- A partir de una lista, generar una nueva lista con los cuadrados de sus elementos

```
lista=[1,2,3,4,5]

lista2=list(map(lambda x:x**2,lista))

print(lista2)
```

Función Map - Ejemplo

- No necesariamente tiene que pasarse siempre una función lambda, puede ser cualquier otra función. Aunque las funciones lambda son ideales
- La función map devuelve un objeto map. Por eso la función list es necesaria para convertir en una lista el resultado.

Función filter

Funciones Filter

- La función **filter** selecciona algunos elementos de una lista para crear una nueva lista con ellos.
- Por ello es indispensable que las funciones que se utilizan en filter deben devolver como resultado **true** o **false**.
- **Los elementos de la lista original que se añaden a la nueva lista son aquellos que devuelven True al aplicarles la función**

Función Filter - Ejemplo

- A partir de una lista de números generar una nueva con solo los impares

```
lista=[1,2,3,4,5]

lista2=list(filter(lambda x:x%2!=0,lista))

print(lista2)
```

Funciones Filter

- Al igual que en **map**, la función **filter** devuelve un objeto del tipo **filter**
- Para poder utilizarlo tenemos que convertirlo en una lista con la función **list**
- **Se pueden utilizar funciones lambda como funciones normales**

Función reduce

Funciones Reduce

- Para poder ser utilizada se tiene que importar el modulo **functools**
- La función **reduce** toma como argumento un conjunto de valores (una lista, tupla o cualquier objeto iterable) y lo “**reduce**” a un único valor.
- Como se obtiene ese único valor a partir de la colección pasada como argumento dependerá de la función aplicada (que debe tener dos parámetros definidos)

Función Reduce - Ejemplo

- Sumar los valores de una lista

```
from functools import reduce

def suma(a,b):
    return a+b

print(reduce(suma,[1,2,3,4]))
```

Como dijimos anteriormente la función pasada como primer argumento debe **tener dos parámetros**. En este ejemplo **reduce** se encargara de llamarla acumulativamente de izquierda a derecha

```
print(suma(suma(suma(suma(1,2),3),4)))
```

Expresiones regulares

Expresiones Regulares

- Son una secuencia de caracteres que define un patrón de búsqueda. Este patrón se utiliza para encontrar coincidencias dentro de **otras cadenas de texto**, basado en delimitadores específicos y regla de sintaxis.
- Por ejemplo el patrón **pa** con alguna de las cadenas que podría coincidir son:
 - **pa**
 - **papa**
 - **ropa**
 - **parpado**

Expresiones Regulares

- Si el patrón se encuentra, decimos que hemos encontrado una coincidencia **match**.
- En el ejemplo anterior utiliza un patrón literal, es decir solo encontramos **coincidencias cuando hay una correspondencia exacta**

Expresiones Regulares

- Las expresiones regulares son patrones que usaremos para encontrar una o varias combinaciones de caracteres en un texto
- Las expresiones regulares se suelen llamar **regex**, son unas secuencias de caracteres que forma un patrón de búsqueda, las cuales son formalizadas por medio de una sintaxis específica
- Dicho de otra forma se pueden interpretar como un conjunto de instrucciones, que luego se ejecutan sobre un texto de entrada para producir un subconjunto o una versión modificada del texto original

Caracteres y meta caracteres

- El patrón va a estar formado por:
 - **Caracteres:** letras, números o signos
 - **Meta caracteres:** no se representan a si mismos, sino que son interpretados de una manera especial. A través de meta caracteres podemos definir diferentes condiciones como agrupaciones, alternativas, comodines, entre otros. Los meta caracteres mas usados son: . * ? + []{}()\$|\

Meta caracteres de posicionamiento

- Los signos **\$** y **^** sirven para indicar donde debe estar situado nuestro patrón dentro de la cadena para considerar que existe una coincidencia. También conocidos como **ancla (anchors)**
 - **^** El patrón debe aparecer al principio de la cadena de caracteres comparada.

Expresión	Coincidencia
^el	el perro de la calle

Meta caracteres de posicionamiento

- \$ El signo \$ indica que el patrón debe aparecer al final del conjunto de caracteres. O mas precisamente antes de un carácter de una nueva linea.

Expresión	Coincidencia
co\$	cuando hace calor me abanico

Si quiero encontrar líneas vacías puede combinar ambas ^\$, porque buscamos una linea que comienza sin nada y termina sin ningún carácter

Escape de caracteres

- Cuando necesitamos incluir en nuestro patrón algún meta carácter como signo literal, es decir, que se interprete por si mismo y no por su función especial.
- Para lograr esto se utiliza el carácter de escape, que es la **barra invertida** \, es decir la barra invertida convierte el carácter en literal
- Por ejemplo si queremos buscar el signo \$ utilizamos \\$

Expresión	Coincidencia
\\$100	El monto total es de \$100

Comodín Punto (.)

- Un punto en el patrón indica cualquier carácter, excepto una nueva linea. Esto significa que puede coincidir con letras, números, espacios en blanco y otros simbolos.

Expresión	Coincidencia
s. I	sal
a...a	aroma
^e.	el

Clases de caracteres []

- Los corchetes definen una clase de caracteres y permiten encontrar cualquiera de los caracteres dentro de un grupo
- Supongamos que queremos encontrar la palabra niño pero tambien si se escribio con n en lugar de ñ

Expresión	Coincidencia
[nñ]	niño
[nñ]	nino

Clases de caracteres []

- La mayoría de los meta caracteres pierden su significado al ser utilizados dentro de clases de caracteres
- Por ejemplo la expresión **[a.]** se refiere literalmente al carácter a y punto
- Una excepción es el carácter ^, que al ser utilizado al inicio indica negación. Por ejemplo: **[^a]** significa cualquier carácter, excepto **a**
- Pero si ^ no se utiliza al principio entonces se toma como un literal. Por ejemplo **[2^]** hace referencia literalmente al símbolo ^

Clases de caracteres: Rangos

- Así como existen caracteres que pierden su significado si se encuentran dentro de corchetes, existe un carácter que solo tiene un significado especial si se encuentra dentro de corchetes
- Este carácter es el **- guion**, se utiliza dentro de corchetes para indicar un rango
- Por ejemplo si quisiésemos referirnos a un **numero hexadecimal** en lugar de [0123456789abcdefABCDEF] podríamos simplificarlo con **[0-9a-fA-F]**

Clases de caracteres: Rangos

Expresión	Coincidencia
201[2-5]	2012 2015
[0-9][0-9]	Su fecha de nacimiento es 17/04/90
[a-zA-Z]	El total del monto es: \$100

Alternativas

- El metacaracter **| (barra vertical)** en expresiones regulares funciona como el operador lógico de alternancia
- Permite especificar múltiples opciones dentro de una misma expresión, de modo que la expresión coincida con cualquiera de las alternancias

Expresión	Coincidencia
[septiembre setiembre]	21 de septiembre 21 de setiembre

Cuantificadores o multiplicadores

- Los multiplicadores permiten establecer cuantas veces puede aparecer un carácter o un grupo de carácter en una cadena que estamos buscando o comparando.
- Se aplican elemento precedente
 - ? indica que el elemento puede aparecer 0 o 1 vez
 - * indica que el elemento puede aparecer 0 o mas veces
 - + indica que el elemento debe aparecer al menos 1 vez, pero puede aparecer mas veces

Cuantificadores o multiplicadores

Expresión	Descripción
col <u>o</u> ?r	Indica que la letra u puede aparecer 0 o 1 vez.
a+b	Cualquier cadena que contenga una o mas letras a seguida por una letra b
h ^{0*}	Cualquier letra h seguida de cero o mas letras o

Cuantificadores o multiplicadores

- En expresiones regulares las **llaves {}** se utilizan para especificar la multiplicidad exacta que es aceptable para un cuantificador.
- Primero se debe indicar el valor mínimo de multiplicidad (opcional si no hay límite inferior) y luego separado por coma el máximo de multiplicidad

Expresión	Coincidencia
$[a-zA-Z]\{2,5\}$	Tiene que ser una secuencia de caracteres mayúsculas o minúsculas entre 2 y 5. Ejemplo Ho UADE

Cuantificadores o multiplicadores

Expresión	Coincidencia
[a-zA-Z]{,2}	<p>Tiene que ser una secuencia de caracteres de mayúsculas o minúsculas hasta 2. Ejemplo Ho UA</p>
[a-zA-Z]{3,}	<p>Tiene que ser una secuencia de caracteres de mayúsculas o minúsculas como mínimo de 3 y sin límite superior. Ejemplo Python</p>

Delimitacion

- Los paréntesis () se utilizan para agrupar subconjuntos de caracteres dentro de una expresión mas grande
- Supongamos que queremos encontrar una secuencia similar a la de un numero de teléfono o patente de un auto

Expresión	Coincidencia
$([0-9]\{4\}-[0-9]\{4\})$	4123-6654
$([A-Z]\{3\}[0-9]\{3\}) \mid ([A-Z]\{2\}[0-9]\{3\}[A-Z]\{2\})$	AF 512 SP

Resumen de la clase

- Métodos interrogativos de String
- Otros métodos de String
- Métodos de formato de string

Ejercitacion

Practica de Cadena de caracteres



Muchas gracias!

Consultas?

