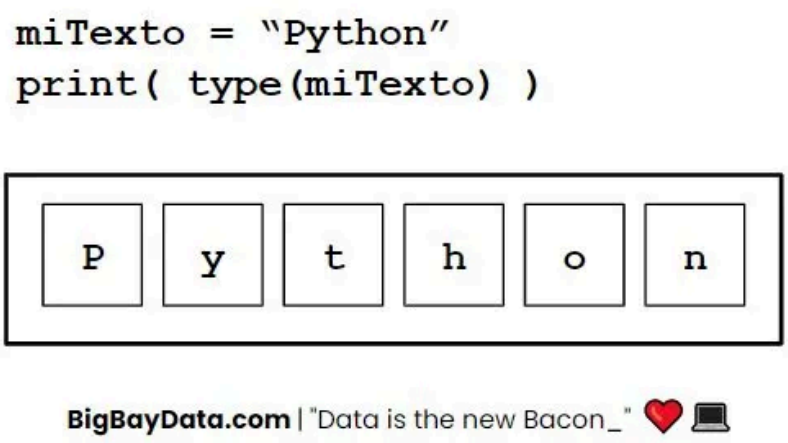
Textos en Python



> Link: https://www.bigbaydata.com/strings-en-python/

Objetivos

- Qué es un texto
- Tipo de dato Str
- Inmutabilidad
- Funciones relevantes
- Errores comunes con Textos

Qué es un Texto: Str

> En inglés, los textos son string o cadenas. Tiene su sentido: el texto `Python` es la suma de `P` + `y` + `t` + `h` + `o` + `n`

In [1]:
```
texto = 'Python' #Recordemos que podemos crearlos con una comilla o 2; pero amba
# texto = "Python" también valdría...
print(texto)
print(type(texto))
```

```
Python
<class 'str'>
```

Tipo de dato STR

Quiero que veas a la variable de antes y pienses en un vagón tren con vagones. Usa el dibujo de arriba:

- El texto comienza con `P`. El primer elemento.
- Después sigue con `y`
- Así hasta terminar con `n`.

Esto es clave. Podemos consultar las posiciones de los elementos de un texto. Para ello, hacemos llamada a la `variable[posicion]`:

In [4]:
```
texto = 'Python'
print(texto[0]) #Todos los elementos empiezan en 0 en programación... Por eso el
print(texto[1])
print(texto[2])
# Podemos acceder al vagón que nos interese en particular.
```

```
P
y
t
```

In [6]:
```
texto = 'Python'
# ¿Qué crees que pasará si llegamos al final de los vagones?
# print(texto[6])
#Python nos dice que no hay nada... Si empezaba por 0 el primer elemento y hay 6
print(texto[5])
```

```
n
```

In [ ]:
```
texto = 'Python'
# Podemos acceder a los vagones de detrás hacia delante...
print(texto[-1]) #Último elemento del vagón... Es decir, la n.

# El penúltimo elemento sería -2...
print(texto[-2]) #¿Qué elemento está accediendo?
```

In [ ]:
```
#podemos invertir una palabra...
texto = 'Palabra'

print(texto[::-1]) #El truco [::-1] es exclusivo de python, invierte la palabra
```

La inmutabilidad

- Redcomendación de uso de print()
- Ejercicio rápido

In [ ]:
```
# Intentamos modificar la cadena
s = "Hola"
s[0] = 'h' #Intentamos acceder al primer vagon y cambiar de valor... Fallará.
```

In [ ]:
```
#Idea rápida para dar solución: Imprimimos vagón a vagón
print('h', s[1], s[2], s[3])

# podemos hacer esto:
s = 'h' + s[1] + s[2] + s[3]
#Otra opción:
s = 'h' + s[1:] #Desde el primer elemento en adelante... (explicado después)
```

Funciones más interesantes

> Tienes las funciones básicas con ejemplos en el artículo. Aquí te dejo con algunas utilidades extra, interesantes:

- len()
- x.upper()
- x.lower()
- x.capitalize()
- x.title()
- x.replace()
- x.strip()
- x.find()
- x.split()
- x.isdigit()

In [2]:
```
s = "Hola, mundo"
print(len(s))  # Salida: 11
# len(s) devuelve el número de caracteres en la cadena 's'
```

```
11
```

In [ ]:
```
s = "Hola, mundo"
print(s.upper())  # Salida: HOLA, MUNDO
# upper() devuelve una nueva cadena con todos los caracteres en mayúsculas
```

In [ ]:
```
s = "Hola, Mundo"
print(s.lower())  # Salida: hola, mundo
# lower() devuelve una nueva cadena con todos los caracteres en minúsculas
```

In [ ]:
```
s = "hola, mundo"
print(s.capitalize())  # Salida: Hola, mundo
# capitalize() devuelve una nueva cadena con el primer carácter en mayúscula
```

In [ ]:
```
s = "hola, mundo"
print(s.title())  # Salida: Hola, Mundo
# title() devuelve una nueva cadena con el primer carácter de cada palabra en ma
```

In [ ]:
```
s = "   Hola, mundo   "
print(s.strip())  # Salida: Hola, mundo
# strip() elimina los espacios en blanco de ambos extremos de la cadena
```

In [ ]:
```
s = "Hola, mundo"
print(s.replace("mundo", "Python"))  # Salida: Hola, Python
# replace() reemplaza todas las apariciones de 'mundo' con 'Python'
```

In [ ]:
```
s = "Hola, mundo"
print(s.find("mundo"))  # Salida: 6
# find() devuelve el índice donde comienza el substring 'mundo'
print(s.find("Python"))  # Salida: -1
# Si el substring no se encuentra, devuelve -1
```

In [ ]:
```
s = "Hola, mundo"
print(s.split())  # Salida: ['Hola,', 'mundo']
# split() divide la cadena en una lista de palabras utilizando el espacio como d
```

```
s2 = "uno,dos,tres"
print(s2.split(','))  # Salida: ['uno', 'dos', 'tres']
# split(',') divide la cadena usando la coma como delimitador
```

In [ ]:
```
s = "12345"
print(s.isdigit())  # Salida: True
# isdigit() devuelve True si todos los caracteres son dígitos

s2 = "123a5"
print(s2.isdigit())  # Salida: False
# isdigit() devuelve False si algún carácter no es un dígito
```

In [ ]:
```
# EXTRA: La concatenación de textos
texto1 = "hola, "
texto2 = "tu nombre"
total = texto1 + texto2 # El simbolo + con textos funcionan como "juntar"; como
```

En big data y en grandes programas informáticos (Software General), los textos son datos relacionados a sensores, operaciones de bancos, posiciones de jugadores, ficheros... y son súper útiles para transformarlos.

Dependiendo de cada caso, necesitaremos utilizar estas funciones de manera conjunta.

Errores más comunes

In [ ]:
```
# 1. Indices fuera de Rango
s = "Hola"
print(s[4])
# Vagones comienzan en 0 siempre. Terminan en N - 1.
```

In [ ]:
```
# 2. El operador +
s = "Hola"
a = 1
print( a + s) #tipos de datos texto y numerico no "suman". El texto usa el + par
```

In [ ]:
```
# 3. Mutabilidad
variable = "Que tal"
variable[1] = 'U' # no podemos cambiar los vagones del tren.
```

Ejercicios Resueltos

In [ ]:
```
#Ej1.

email = 'misupergmail@gmail-com,otrocorreo@gmail-com,unacuentamas@gmail-com'

# Separar los correos electrónicos utilizando la coma como delimitador
emails_separados = email.split(',')

# Lista para almacenar los correos electrónicos corregidos
emails_corregidos = []

# Reemplazar las ocurrencias de '-com' por '.com' en cada correo electrónico
for correo in emails_separados:
```

```
    correo_corregido = correo.replace('-com', '.com')
    emails_corregidos.append(correo_corregido)

# Imprimir los correos electrónicos corregidos
for correo in emails_corregidos:
    print(correo)
```

In [1]:
```
#Ej2.

# Lista de notas
notas = '1, 4.3, 7.1, 4.6, 5.1, 6.6, 7.2, 8.8, 10, 9.8, 7.6'

# Convertir la cadena de notas en una lista de números
notas_lista = []

for nota in notas.split(','):
    nota_limpiada = nota.strip()  # Eliminar espacios en blanco alrededor de la
    nota_float = float(nota_limpiada)  # Convertir la nota a tipo flotante
    notas_lista.append(nota_float)  # Agregar la nota a la lista

# Imprimir la lista de notas transformada
print(notas_lista)
```

```
[1.0, 4.3, 7.1, 4.6, 5.1, 6.6, 7.2, 8.8, 10.0, 9.8, 7.6]
```

In [2]:
```
#Ej3.
#Opción 1
datos = 'aaaaaa1.2b2cde110230'
datos = datos.replace('a','')
print(datos)
datos = datos.replace('b','')
print(datos)
datos = datos.replace('c','')
print(datos)
datos = datos.replace('d','')
print(datos)
datos = datos.replace('e','')
print(datos)
print('Fin Opción 1')
# Opción 2
datos = datos.replace('a','').replace('b','').replace('c','').replace('d','').re

print(datos)
```

```
1.2b2cde110230
1.22cde110230
1.22de110230
1.22e110230
1.22110230
Fin Opción 1
1.22110230
```

In [ ]:
```
#Ej4.
# Solicitar al usuario introducir una fecha
fecha = input("Introduce una fecha en formato dd/mm/aaaa: ") #Ej: 10/05/1990

# Separar el año de la fecha introducida
fecha = fecha.split('/') #Recuerda que split localiza el símbolo y trocea...
#Siguiendo el Ej: split de '/' a la fecha 10/05/1990 devuelve una lista [10, 05,
dia = fecha[0] #primer bloque: 10
```

```
mes = fecha[1] #segundo bloque: 05
anyo = fecha[2] #tercer bloque: 1995

# Determinar si el año es bisiesto o no
if (anyo % 4 == 0 and anyo % 100 != 0) or (anyo % 400 == 0): #código optimizado.
    print("El año",anyo," es bisiesto.")
else:
    print("El año",anyo," no es bisiesto.")
```

In [ ]:
```
#Ej5.

# Solicitar al usuario que ingrese un apellido
apellido = input("Ingresa un apellido: ")

# Convertir el apellido a minúsculas para ignorar diferencias entre mayúsculas y
apellido = apellido.lower()

# Verificar si el apellido es palíndromo
if apellido == apellido[::-1]:
    print("El apellido",apellido,"es palíndromo.")
else:
    print("El apellido",apellido,"no es palíndromo.")
```