# Clase01_1_Cadenas

March 13, 2024

## 1 Seminario de Lenguajes - Python

### 1.1 Temario

- Cadenas de caracteres
- Función range()

## 2 Tipos de datos

### 2.1 ¿Qué nos indica un tipo de datos?

- El tipo de datos nos indica **qué valores** y **qué operaciones** podemos hacer con una determinada variable.

## 3 ¿Qué tipos de datos vimos en la clase?

- Números: **int** y **float**
- Booleanos: **bool** (que mencionamos que eran números también)
- Cadenas de caracteres: **str**

La función **type()** nos permite saber de qué tipo es un determinado objeto referenciado por una variable.

[ ]:
```
x = True
type(x)
```

## 4 También vimos que hay algunas conversiones de tipo implícitas y otras explícitas

[ ]:
```
half = 10 / 2
type(half)
```

[ ]:
```
half = int(10 / 2)
type(half)
```

## 5 Las cadenas de caracteres

- Secuencia de caracteres encerrados entre comillas simples ' ' o comillas dobles " ".
- También se pueden definir con """ """.

[ ]:
```
error_mensage = "ATENCION: la opción ingresada no es correcta."
error_mensage
```

[ ]:
```
zen = """ The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
.......
        """
print(zen)
```

## 6 Operaciones con cadenas de caracteres

- Concatenación: +
- Repetición: *
- Longitud de la cadena: **len()**

[ ]:
```
part_1 = "Python "
part_2 = "es lo más!"
print(part_1 + part_2)
print(part_1 * 5)
print(len(part_1))
```

## 7 Algo más sobre cadenas de caracteres

- Cada elemento de la cadena se accede mediante un índice entre []

[ ]:
```
word = "Python"
word[-2]
```

- El índice puede ser negativo.

## 8 Subcadenas - slicing

[ ]:
```
#word[3:]
word[:]
```

- El operador **:** permite obtener subcadenas. Esto se denomina **slicing**.
- El formato es **cadena[inicio:fin]**
- NO incluye al elemento cuyo índice es **fin**.
- **[:]** devuelve toda la cadena.

- **Si los índices son negativos, se recorre de derecha a izquierda.**

## 9 Probemos esto:

[ ]:
```
word[1] = 'm'
```

- **Las cadenas son INMUTABLES.**

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

#### 9.0.1 Tenemos que acostumbrarnos a leer los mensajes de error.

## 10 Algo más sobre cadenas de caracteres

- Ya mencionamos que en Python, todos son objetos.
- Si bien retornaremos a esto más adelante, podemos mencionar que los objetos tienen **propiedades y métodos**.
  - objeto.propiedad
  - objeto.metodo()
- Volviendo a las cadenas, algunos métodos que podemos utilizar son:

[ ]:
```
sentence = "Python es lo más!"
#sentence.upper()
sentence.lower()
```

[ ]:
```
sentence.islower()
#sentence.isupper()
```

### 10.1 Algo un poco más interesante:

[ ]:
```
sentence = "Somos campeones del mundo!!!!!!"
sentence.count("!")
```

[ ]:
```
sentence.center(70, "*")
```

[ ]:
```
"   Somos campeones del mundo!!!!!    ".strip()
```

## 11 Y un poco más...

[ ]:
```
word = "_caminar"
#word.startswith("_")
word.endswith(("ar", "er", "ir"))
```

Podemos plantear otra solución al DESAFÍO 6 planteado en clase.

[ ]:
```
# Solución
```

## 12 El método split()

```
[ ]: "Somos campeones del  mundo!!!".split()
```

Probar: ¿de qué tipo es el objeto retornado por split?

- +Info

## 13 El operador in

- Este operador retorna True o False de acuerdo a si un elemento está en una colección o no.
- Como las cadenas de caracteres son **secuencias de caracteres** por lo que puede utilizarse este operador.

```
[ ]: word = input("Ingresá una palabra: ")
if "a" in word:
    print("Hay letras a.")
else:
    print("No hay letras a. ")
```

## 14 El módulo string

- Python tiene un módulo denominado string que contiene mucha funcionalidad para la manipulación de cadenas.
- Para acceder a esta funcionalidad hay que **importarla**. Esto lo veremos en detalle más adelante.

```
[ ]: import string
letters = string.ascii_letters
let_minus = string.ascii_lowercase
num_digits = string.digits

letters
```

Ahora podemos tener otra solución al DESAFÍO 3 planteado en clase: > Dado una letra ingresada por el teclado, queremos saber si es mayúscula o minúscula.

```
[ ]: import string
let_lower = string.ascii_lowercase
let_upper = string.ascii_uppercase

letter = input("Ingresar una letra: ")
if letter in let_lower:
    print("Es minuscula.")
elif letter in let_upper:
    print("Es mayúscula.")
else:
    print("No es una letra.")
```

## 15 Cadenas con formato

- Es posible definir cadenas con determinados formatos utilizando el método **format**.
- La forma general es:

`var_str.format(argumentos)`

- Observemos los siguientes ejemplos:

[ ]:
```
tries = 5
print('Hola {} !!! Ganaste! y necesitaste {} intentos!!!!'.format("Lionel",␣
↪tries))
```

[ ]:
```
for number in "123":
    x = int(number)
    print("{0:2d} {1:3d} {2:4d}".format(x, x*x, x*x*x))
```

## 16 Los f-String

- Estuvimos usándolos en los ejemplos donde se mostraba el contenido de una variable además de texto.

[ ]:
```
data = 21
print(data)
data ='hola!'
print(f'{data} ¿Cómo están?')
```

- Fueron introducidos a partir de la versión 3.6.
- Ver la PEP 498
- +Info en la documentación oficial
- Es una forma más sencilla de usar el format y que tiene más funcionalidades.

## 17 Un ejemplo

[ ]:
```
tries = 5
the_goat = "Lionel"
print(f'Hola {the_goat} !!! Ganaste! y necesitaste {tries} intentos!!!!')
x = 4
print(f"{x:2d} {x*x:3d} {x*x*x:4d}")
```

## 18 Algunas cosas interesantes

[ ]:
```
sentence_1 = "En Argentina nací"
sentence_2 = "Tierra del Diego y Lionel"
sentence_3 = "De los pibes de Malvinas"
sentence_4 = "Que jamás olvidaré."
```

```
print(sentence_1)
print(sentence_2)
print(sentence_3)
print(sentence_4)
```

[ ]:
```
print(f"La mejor canción de todas:\n{sentence_1:<30}\n{sentence_2:>50}")
print(f"\n{sentence_3:^30}")
print(f"\n{sentence_4:*^50}")
```

### 18.1 Permite simplificar para "debuggear" de forma simple

Suponemos que queremos saber el contenido de dos variables:

[ ]:
```
x = 10
y = 25
print(f"x = {x} , y = {y}")
```

¿Cómo nos ayuda f'string para simplificar?

[ ]:
```
print(f"{x = }, {y = }")
```

Y si queremos agregarle decimales

[ ]:
```
print(f"{x/y = :.3f}, {y = }")
```

## 19 Un artículo sobre sistemas de codificación

-Unicode & Character Encodings in Python: A Painless Guide

## 20 DESAFÍO

> Escribir un programa que ingrese 4 palabras desde el teclado e imprima aquellas que contienen la letra "r".

**Pensar:** ¿podemos usar la instrucción **for** tal como la usamos hasta ahora para generar las 4 iteraciones?

- La sentencia **for** permite iterar sobre una **secuencia**.

```
for variable in secuencia:
        instrucción
        instrucción
        ...
        instrucción
```

[ ]:
```
string_num = "0123"
for elem in string_num:
    print(elem)
```

## 21 Alguien podría pensar en plantear esto:

[ ]:
```
for i in "1234":
    word = input("Ingresá una palabra: ")
    if "r" in word:
        print(word)
```

Pero.. ¿sería una solución correcta? ¿Qué pasa si queremos ingresar 200 palabras?¿O 2000?

## 22 La función range()

- Esta función devuelve una secuencia de números enteros.
- Puede tener de 1 a 3 argumentos:

`range(valor_inicial, valor_final, paso)`

- Es posible invocarla con uno, dos o los tres argumentos.

[ ]:
```
for i in range(4, 23, 3):
    print(f"{i}", end="-")
```

## 23 Entonces, una mejor forma sería:

[ ]:
```
 for i in range(4):
    word = input("Ingresa una palabra: ")
    if "r" in word:
        print(word)
```