

title: Presentacion Python 2019
Author: Claudia Banchoff, Viviana Harari
description: Clase 2
keywords: Cadenas, Listas, Tuplas, Conjuntos y Sentencia for
css: estilo.css

Seminario de Lenguajes - Python

Cursada 2019

Temario

- Repaso clase anterior.
 - Características generales del lenguaje.
 - Variables y tipos de datos básicos
 - Estructuras de control: if y while
 - Tipos de datos estructurados
 - Manejo de cadenas.
 - Listas
 - Tuplas
 - Conjuntos
 - Estructura de control
 - for
-

Repaso: Características generales del lenguaje

- Es software libre.
 - Es interpretado.
 - Formas de ejecución.
 - Usando el modo interactivo obteniendo una respuesta por cada línea. Sesión interactiva o
 - Escribiendo el código de un programa en un archivo de texto y luego ejecutándolo.
 - Tipado dinámico y fuerte.
 - Sintaxis sensible a las mayúsculas y minúsculas.
-

Repaso: Variables y Tipos de Datos

- Las variables
 - NO se declaran. Se crean cuando se les asigna un valor.
 - Son case sensitive
 - Tipos vistos:
 - Números: Enteros, Flotantes.
 - Booleanos
 - Cadenas. Definición.
-

Repaso: Estructuras de Control

- Estructuras de control.
 - Condicional: sentencia **if**
 - Iteración: sentencia **while**
-

Repaso: ¿Recordamos el juego que programamos la clase anterior?

- "Adivina que número piensa la compu..."
-

```
#Utilizo una función que genera números aleatorios en un cierto rango
import random
numero_compu= random.randrange(100)

#Inicializo la variable que cuenta la cantidad de oportunidades y comienzo
#con el juego

cont=1
while cont < 11:
    #Pido ingresar el número al usuario
    ingresa_numero= int(input('Ingresa el número que pensó la compu en un rango de 0 a 100'))

    #Evalúo si es le número generado por la computadora
    if ingresa_numero == numero_compu:
        print ('Ganaste! y lo hiciste en', cont, 'intentos!')
        cont= 13
    else:
        print ('No.. ese número no es... Sigue pensando..')
        cont= cont + 1
```

```
#Consulta si uso todos los intentos..  
if cont == 11:  
    print ( '\n Perdiste :(\n La compu pensó en el número:', numero_compu)
```

¿Modificamos el programa?

Pensemos en cómo modificar el programa para que ofrezca, al jugador, **ayudas** a través de "**pistas**".

Una posibilidad podría ser..

- Programa
-

Más sobre cadenas de caracteres

- Accediendo a los caracteres de las cadenas

```
cadena = 'Hola que tal'
```

0	1	2	3	4	5	6	7	8	9	10	11
H	o	l	a		q	u	e		t	a	l

- **IMPORTANTE:** los índices comienzan en 0.

```
print(cadena[0])  
print(cadena[len(cadena)])
```

¿Qué les parece que imprime?

Más sobre cadenas

```
cadena = 'La casa es grande'  
print(cadena[5:8])  
print(cadena[9:-1])
```

- ¿Qué les parece que imprime?
-

Más sobre cadenas

- 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.

¿Se puede hacer `cadena[5] = 'm'`?

Más sobre cadenas

- Las cadenas son INMUTABLES
 - Hay muchas funciones que se aplican a cadenas como ser:
 - **upper() lower() capitalize()**
 - **find() index() ...**
-

Tipos Estructurados - Listas

- Una **lista** es una colección ordenada de elementos.
- Es heterogénea, es decir que pueden contener cualquier tipo de datos, inclusive listas.

```
lis = [ 22, True, 'una lista', [1,7] ]
```

Acceso a los elementos de una lista

- Se accede a través de un índice que indica la posición del elemento dentro de la lista encerrado entre corchetes `[]`.
- **IMPORTANTE:** Al igual que las cadenas los índices comienzan en 0.
- Algunas operaciones:

```
lis = [ 22, True, 'una lista', [1,7] ]  
print( lis[3][1] )  
print( lis[-3] )
```

¿Se puede hacer..?

```
lis[1] = False
```

Manejo de listas

- Las listas son MUTABLES
- Al igual que en el caso de las cadenas de caracteres, se puede obtener una porción de una lista usando el operador :

```
lis = [ 22, True, 'una lista', [1,7] ]  
print( lis[1:3] )  
print( lis[ :2] )  
print( lis[2:] )
```

- Si no se pone inicio o fin, se toma por defecto las posiciones de inicio y fin de la lista.
-

Manejo de listas

- Diferencia entre hacer `lis2 = lis` y `lis2 = lis[:]`
 - **lis2 = lis1** hace que el lis2 apunte a lis1 (ambos apuntan a la misma zona de memoria)
 - **lis2 = lis[:]** hace que dos direcciones de memoria tengan el mismo contenido
- Uso de método **copy** en Python 3.

```
lista = [ 22, True, 'una lista', [1,7] ]  
lis1 = lista.copy()
```

Manejo de listas

- Operadores
- lis1 = [22, True, 'una lista', [1,7]]**
- **+**: Para concatenar listas
lis2 = lis1 + [9,8,7] => lis2 quedaría: [22, True, 'una lista', [1,7], 9, 8, 7]
 - *****: Para repetir las listas
lis2 = lis1 * 2 => lis2 quedaría: [22, True, 'una lista', [1,7], 22, True, 'una lista', [1,7]]
-

Manejo de listas

- Analicemos el siguiente código...

```
lista = [[1 , 2]] * 3
lista [0][1] = 'cambio'
print (lista)
```

- ¿Imprime [[1, 'cambio'], [1, 2], [1, 2]] o [[1, 'cambio'], [1, 'cambio'], [1, 'cambio']]?. ¿Por qué?.
- Y si hacemos lista = [[1 , 2], [1 , 2], [1 , 2]]. ¿Sucede lo mismo?

Note

Aclarar que imprime la segunda opción porque el * repite la misma lista, con lo cuál queda almacenado punteros a las mismas No sucede lo mismo, en el segundo caso los elementos son todos diferentes, en el primer caso es el mismo objeto referenciado 3 veces

Manejo de listas

- Algunos métodos o funciones aplicables a listas
extend() append() index()
remove() pop() count() etc.

Retomamos estructuras de control

Antes de continuar con las estructuras de datos restantes vemos la última estructura de control

Sentencia for .. in

- Estructura que permite iterar sobre una secuencia.
- Sirven para recorrer una secuencia, como ser una lista, una tupla, etc.

```
for variable in serie_de_valores:
    acción
    acción
    ...
    acción
```

- La primer línea se leería como: “Para cada elemento de la secuencia.. se debe realizar ..”
-

Sentencia for .. in

Ejemplo

```
dias = ["domingo", "lunes", "martes", "miércoles", "jueves", "viernes", "sábado"]
for d in dias:
    print(d)
```

Funcion range()

- Esta función devuelve una lista de números enteros.
 - Puede tender de 1 a 3 argumentos:
 - 1 argumento: **range(5)** => devuelve [0,1,2,3,4]
 - Comienza con 0 hasta el argumento - 1
 - 2 argumentos: **range(2,5)** => devuelve [2,3,4]
 - Comienza con el argumento1 hasta el argumento2 - 1
 - 3 argumentos: **range(2,6,2)** => devuelve [2,4]
 - Comienza con el argumento1 hasta el argumento2 - 1, pero con un incremento de 2
-

Simplificando un proceso

```
i = valor inicial
while i <= valor final:
    acciones
    i += 1
```

- El mismo se puede reemplazar con un for utilizando range()

```
for i in range (valor incial, final + 1):
    acciones
```

Sentencia for .. in

- Ejemplo del for de la forma que lo usan los lenguajes como C, Java, Pascal, etc. dónde se itera sobre las posiciones y no sobre los elementos. **Condición:** uso de la función range()

```
dias = ["domingo", "lunes", "martes", "miércoles", "jueves", "viernes", "sábado"]
for d in range(7):
    print("Elemento ", d, "de la lista es: ", dias[d])
```

¿Implementamos un nuevo juego?

- Con todo lo visto hasta ahora se podría implementar otro juego, por ejemplo:
- El juego del **"ahorcado"**, con:
 - una sola lista de palabras y
 - con tres posibilidades de perder
- ¿Lo vemos?.

Juego del ahorcado

- Programa del [ahorcado](#)

Tipos estructurados: tuplas

- Son colecciones de datos ordenados.
- Se definen de la siguiente manera:

```
tupla1 = 1,2
tupla1 = (1,2)
```

- Tupla de un elemento: `tupla1=(1,)` , caso contrario `tupla1=(1)` sería un entero.
- Tupla de ningún elemento: `tupla1=()`

Tuplas vs. Listas

```
tupla1 = (1, 2)
```

- Similitud con las listas:
 - Formas de acceder a sus elementos, uso de `[]`
 - `tupla1[1]` => devuelve 2
- Diferencias con las listas:

- Son **INMUTABLES**, su tamaño y los valores de las mismas NO pueden cambiar

```
tupla1 = (1, 2)
tupla[1] = 10 #Esto daría error!!
```

- ¿Qué pasó?
-

Obteniendo subtuplas

```
t = (1, 2, 3, "hola")
print(t[1:4])
t_nueva = ("nueva",) + t[1:3]
print(t_nueva)
```

- ¿Qué sucede si hubiésemos puesto?

```
t_nueva=( 'nueva' ) + t[1:3]
```

Tipos Estructurados - Set

- Un conjunto es una colección de datos de igual o diferente tipo, **desordenada** y **sin elementos duplicados**.
 - Operaciones sobre los tipos conjuntos:
 - Saber si un elemento pertenece o no al conjunto;
 - Operaciones matemáticas como la unión, intersección, diferencia y diferencia simétrica.
-

Creando conjuntos

```
c1 = set((1, 2, 3, 4))
c2 = set("alabanza")
```

- ¿c1?, ¿c2?
 - ¿Qué pasa con las "a" que se repiten?
-

Operaciones sobre conjuntos

- **in** permite indagar si un elemento pertenece o no a un conjunto.

- `|` permite la unión entre dos conjuntos.
- `&` permite la intersección entre dos conjuntos. En este caso da vacío.

```
colores_calidos = set(["celeste", "rosa"])
colores_fuertes = set(["azul", "rojo"])
print("azul" in colores_calidos)
print("azul" in colores_fuertes)

todos = colores_calidos | colores_fuertes
x = colores_calidos & colores_fuertes
```

- ¿todos? ¿x?
-

Operaciones sobre conjuntos

- Para copiar conjuntos podemos usar la sentencia de asignación o usando **`copy()`**.
- Se puede adicionar un elemento con **`add()`**.
- Se puede eliminar un elemento del conjunto con **`discard()`**.

```
conj1 = set([1, 2, 3, 4])
conj2 = conj1

conj3 = conj1.copy()
conj1.add(7)
conj1.discard(1)
```

- ¿Qué diferencias hay entre utilizar **`copy()`** o `=` para asignar dos conjuntos?
-

¿Qué vimos hoy?

Resumen

- Estructuras de datos: cadenas, listas, tuplas y conjuntos.
- Estructuras de control de iteración: `for`
- La función `range()`.