title: Presentacion Python 2019
Author: Claudia Banchoff, Viviana Harari
description: Clase 3
keywords: Diccionarios y Funciones
css: estilo.css

Seminario de Lenguajes - Python

Cursada 2019

Temario

- Repaso clase anterior.
    - Manejo de cadenas
    - Tipos de datos estructurados: Listas, Tuplas y Conjuntos
    - Estructura de control: for
- Tipos de datos estructurados.
    - Diccionarios
    - Funciones

Repaso: Cadenas

```
cadena = "Estamos repasando Python"
print (cadena[4])
nueva_cadena = cadena[17:]
```

- Tipo de dato INMUTABLE
- Múltiples funciones sobre cadenas

Repaso: Listas

- Colección ordenada
- **Puede contener cualquier tipo de datos**, inclusive listas.
- Tipo de dato MUTABLE

```
lis = [1, "a", True, [1,7]]
lis[3] = 21
lis[-2] = 20
print(lis[1:3])
print(lis[:2])
```

- ¿Diferencias entre hacer lis2=lis1 y lis2=lis1[:]?

Repaso: Tuplas

- Colección ordenada
- **Puede contener cualquier tipo de datos**, inclusive tuplas.
- Tipo de dato INMUTABLE

```
tupla = [False, "hola",[10,7,8], (1,2)]
print(tupla[1:3])
print(tupla[:-2])
```

- ¿Se puede hacer tupla[1] = 'chau' ?

Repaso: Conjuntos

- Colección no ordenada y sin elementos repetidos
- Operaciones sobre conjuntos: | y &
- Se pueden quitar o agregar elementos al conjunto

```
conjunto_1= {1,2,3}
conjunto_2= set((8,9,7))
```

Repaso: Sentencia For

- Estructura de control de repetición

```
for letra in ('repasando'):
        print (letra)
```

- Uso de la funcion range ()

Tipos Estructurados - Diccionarios

- Un diccionario es un conjunto **no ordenado** de pares de datos: **clave:valor**.
- Las **claves** deben ser **únicas** en un diccionario.
- Se definen con **{ }**.

```
alumnos = {1:"Juan",2:"Julia", 3:"Carlos"}
```

- **{ }** => Define un diccionario vacío.
- Las claves pueden ser cualquier tipo inmutable;
    - Las cadenas y números siempre pueden ser claves.
    - Las tuplas se pueden usar sólo si no tienen objetos mutables.

Las claves de un diccionario

```
dic1 = {(1, 2): 'Hola', 2: 'chau', (2, 3): 'otra vez'}
dic2 = {[1,2]: "Hola", 2: 'chau', (2, 3): 'otra vez'}
```

- ¿Qué diferencias hay entre dic1 y dic 2?

Acceder a elementos de un diccionario

```
dic1 = {1: 'uno', 2: 'dos', 3: 'tres'}
meses = {"enero": 31, "febrero": 28, "marzo": 31}

elemento = dic1[1]
dias  = meses["enero"]
```

- Para extraer un valor, se accede mendiante la clave usando **[ ]**. Es un error extraer un valor usando una clave no existente.

Guardar elementos en un diccionario

```
meses = {"enero": 31, "febrero": 28, "marzo": 31}

meses["febrero"] = 29
meses["abril"] = 30
```

- Si se usa una clave que ya está en uso para guardar un valor, el valor que estaba asociado con esa clave se pierde, si no está la clave, se agrega

Copiando diccionarios

```
meses = {"enero": 31, "febrero": 28, "marzo": 31}
meses1 = meses
meses2 = meses.copy()
```

- ¿Diferencias?

Eliminar elementos de un diccionario

- **del** permite borrar un par clave:valor
- **clear()** permite borrar todo

```
meses = {"enero": 31, "febrero": 28, "marzo": 31}
del meses["febrero"]
meses.clear()
```

- **len(nomDic)** => Devuelve la cantidad de elementos del diccionario

---

Otras operaciones

- **keys():** devuelve una lista de todas las claves en uso del diccionario.
- **in:** verificar si una clave está en el diccionario.
- Otras métodos: **values()** y **items()**

```
meses = {"enero": 31, "febrero": 28, "marzo": 31}
nombres = meses.keys()
meses.values()
meses.items()
```

El constructor

- **dict()** permite crear diccionarios.
- Se denomia "constructor" y crea un diccionario directamente desde listas de pares clave-valor guardados como tuplas.

```
dict([(1,"Juan"), (2,"Julia"), (3,"Carlos")])
{1:"Juan", 2:"Julia", 3:"Carlos"}
```

- Cuando los pares siguen un patrón, se puede especificar de forma compacta la lista de pares clave-valor por comprensión.

```
dict ([(x, x**2) for x in (2, 4, 6)])
{2: 4, 4: 16, 6: 36}
```

¿Modificamos el juego del ahorcado?

- Pensemos en modificar el juego del ahorcado para que el jugador pueda elegir por temas.
- Supongamos los siguientes temas:
    - colores
    - animales
    - comidas
- A pensar.....

Modificación posible

Modificación

```
import random

#Preparo el juego

#Defino conjunto de palabras a trabajar por temas
palabras = {1:['gato', 'perro','pato','elefante','lobo'], 2:['rojo','azul','verde','amar

#Defino estructura del ahoracado
ahorcado = [' O ', '/|\\','/ \\']

#Comienza el proceso del juego
#Pido que el jugador elija un tema
tema = int(input('Elige un tema:\n 1: animales\n 2: colores\n 3: comidas\n '))

#Selecciono la palabra a trabajar
pal = palabras[tema][random.randrange(len(palabras[tema]))]
```

Tipos de datos estructurados

Se puduieron ver los diferentes tipos de datos estructurados que tiene Python y las mejoras que produjeron en el desarrollo del juego "el ahorcado"

¿Vemos ahora funciones?

Pensemos en el siguiente problema ...

---

class: destacado

Guido ingresa en el bosque de los árboles mágicos.

Algunos árboles al tocarlos dejan caer hojas de oro, y otros, al tocarlos te convierten en piedra.

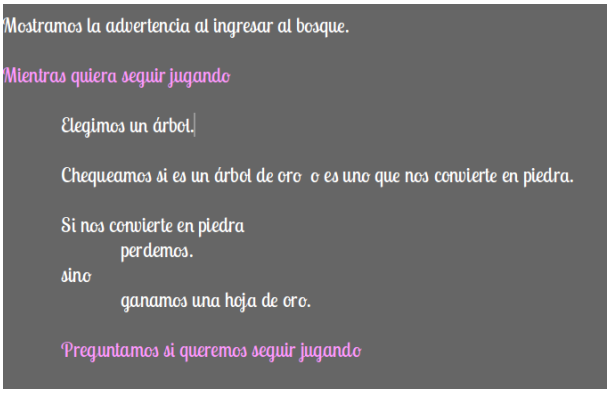
Sin nada que los diferencien, se debe arriesgar ...

Una solución

```
Mostramos la advertencia al ingresar al bosque.

Elegimos un árbol.

Chequeamos si es un árbol de oro  o es uno que nos convierte en piedra.

Si nos convierte en piedra
        perdemos.
sino
        ganamos una hoja de oro.
```

¿Y si queremos repetir el juego hasta que me canse de jugar?

Una solución



¿Podemos pensar en dividir nuestro programa en procesos?

Modularizando nuestros programas

- En Python, usamos **funciones** para modularizar nuestro código.
- Las funciones pueden recibir parámetros.
- Y también retornan siempre un valor. Esto puede hacerse en forma implícita o explícita usando la sentencia **return**.

Funciones en Python

- Ya usamos funciones: ¿cuáles?

Funciones en Python

- Usamos:
    - float(), int(), str() y long()
    - len(), ord(), chr()
    - range(), randrange()
    - sorted()

Funciones en Python

- Es posible definir nuestras propias funciones.
- Forma general:

```
def nombre_funcion(parametros):
        sentencias
        return <expresion>
```

- *Recuerden:* el cuerpo de la función debe estar **indentado**.

Una solución al juego planteado

- Veamos el código: bosque.py
- ¿Algunas mejoras?

Analizamos el código

- La función **elegimos_arbol** retorna explícitamente un valor (el árbol elegido).
- Pero las otras dos funciones no retornan nada. ¿Es así?

```
def advertencia():
    print('''Estás entrando al bosque encantado. En este bosque hay árboles mágicos.
    ¡Cuidado al tocarlos!
    Algunos pueden hacerte rico o convertirte en piedra.
    Probá tu suerte.''')
    print()
```

- ¿Qué valor retorna una función que no tiene una sentencia **return**?
- ¿Cómo lo probamos?

Seguimos analizando el código

- Veamos la función **chequeamos_arbol**

```
def chequeamos_arbol(arbol_elegido):
    print('Has tocado el arbol {0:1s}'.format(arbol_elegido))
    time.sleep(2)
    print('Veamos cuál es tu suerte...')
    print()
    time.sleep(2)
```

```
    nuestra_suerte = random.randint(1, 2)

    if arbol_elegido == str(nuestra_suerte):
         print('¡Hoy es tu dia! ¡Ganaste una hoja de oro! ')
    else:
         print('Mmmm creo que hoy no es tu día :( ')
```

- Esta función tiene un parámetro.
- ¿Hay distintas formas de pasar parámetros en Python?
- ¿Cómo podemos probar esto?

Parámetros en Python

- Pensemos en el ejemplo más simple:

```
def modifico_parametro(x):
        x = 10


...
a = 2
modifico_parametro(a)
print(a)
```

- ¿Cómo está pasado el parámetro a la función?

---

class: destacado

Parámetros en Python

Los argumentos en Python son **una copia de la referencia** al objeto pasado.

- ¿Importará el tipo del parámetro?
- ¿Es lo mismo pasar un número entero que una lista?

Parámetros en Python

```
def modifico_entero(x):
        x = x + 1

def modifico_lista(x):
        x[0] = "cero"

num = 2
lista = [1, 20]

modifico_entero(num)
modifico_lista(lista)
print(num, lista)
```

- ¿Qué observamos?

Parámetros en Python

```
def modifico_lista(x):
        y = x[:]
        y[0] = "cero"

lista = [1,20]
modifico_lista(lista)

print(lista)
```

- ¿Y en este caso?
- Y si invoco con **modifico_lista(lista[:])** sería necesario la primer asignación? ¿Les parece buena esta práctica?

Parámetros en Python

```
def modifico_lista(x):
        x[0] = "cero"

tupla = (1,20)
modifico_lista(tupla)

print(tupla)
```

- ¿Qué pasa con este código?

Tarea para el hogar ...

El ahorcado con funciones

¿Qué vimos hoy?

Resumen

- Estructuras de datos complejas:
    - Listas
    - Tuplas
    - Conjuntos
    - Diccionarios.
- Diferencias y semejanzas entre las mismas.
- Funciones
    - Definición
    - Pasaje de parámetros