title: Presentacion Python 2019
Author: Claudia Banchoff, Viviana Harari
description: Clase 6
keywords: Archivos
css: estilo.css

Seminario de Lenguajes - Python

Cursada 2019

Antes de iniciar la clase...

Hablemos de la evaluación pasada

Temario

- Repaso de la clase anterior.
- Manejo de archivos.
    - Distintos tipos de archivos.

Repaso: por Python Plus!!

Repaso: por Python Plus!!

- ¿Qué significa que un argumento de una función tenga un valor por defecto?
- ¿Qué está mal en el siguiente código?

```
def funcion(par1=10, par2=11, par3):
        sentencias...

funcion("hola")
```

Repaso: por Python Plus!!

- ¿Qué diferencias hay entre las siguientes definiciones?

```
def funcion1(*args):
        sentencias...

def funcion2(**kargs):
        sentencias
```

- ¿Cómo las puedo invocar?

Repaso: por Python Plus!!

- ¿Qué es un espacio de nombres?
- ¿Qué relación tiene con la sentencia **import**?
- ¿Qué diferencias hay entre **import** y **from import**?
- ¿Qué significa que la importación se realización **sólo una vez por sesión del intérprete**?
- ¿En qué ocasiones esta sentencia puede provocar un error?

```
from mi_modulo import *
```

- Veamos un ejemplo concreto: funciones.py y uso_modulos.py de la clase pasada.
- ¿Qué contiene la variable PYTHONPATH?¿Cómo la accedo?

Repaso: por Python Plus!!

- ¿Qué significa este código?

```
if __name__ == "__main__":
    import sys
    vocales(sys.argv[1])
```

- ¿A qué hace referencia __**name**__?¿Y "__main__"?
- ¿Y **sys.argv[1]**?

Repaso: por Python Plus!!

- ¿Qué diferencias y semejanzas hay entre el módulo random y el módulo pattern?
- ¿A qué se denomina **paquete** en Python?

Formateando cadenas

- Ya usamos **cadena.format()**.
- Observemos estos ejemplos:

```
print('Has tocado el arbol {0:1s}'.format(arbol_elegido))

for x in range(1,11):
        print("{0:2d} {1:3d} {2:4d}".format(x, x*x, x*x*x))

print('\n{:<30}\n{:<30}\n{:<30}'.format('alinzquierda', 'xxxxx', 'yyyyyyyy'))
print('\n{0:>30}\n{0:>30}'.format('alineado a derecha'))
print('\n{:^30}'.format('centrado'))
print('\n{:*^30}'.format('relleno con *'))
```

Formateando cadenas

- **cadena.format(*args, **kwargs)**: retorna una copia de cadena con los cambios especificados en formato aplicados.
- Ver más ejemplos en la documentación oficial de format
- Otros ejemplos:

```
print("\n Hola {0}!! \n\nHoy es  {dia} de {mes}".format("chicos", dia=29, mes="marzo"))
contactos = {"Juan":12345, "Pedro":54321, "Maria":12121212}
for nom, celu in contactos.items():
        print("{0:15}==> {1:10d}".format(nom, celu))
```

Pensemos en las siguientes situaciones

¿Qué estructura usamos si queremos:

- guardar los puntajes cada vez que jugamos a un juego determinado?,
- tener un banco de preguntas para que cada vez que juguemos al juego de repaso las pueda acotar por temas?,
- manipular los Python Plus de los estudiantes por turnos?.

¿Qué tienen todas estas situaciones en común?

---

class: destacado

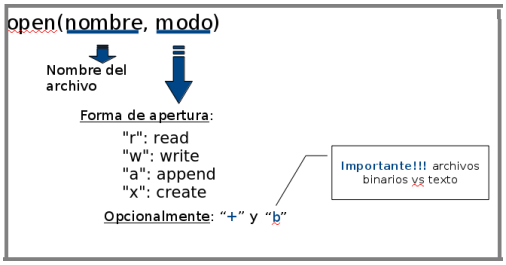
Necesitamos una estructura que permita que los datos puedan **persistir** cuando la ejecución del programa finalice.

Algunas consideraciones antes de empezar

- Lo básico: ¿qué es un **archivo**?
- ¿Cómo podemos manipular los archivos desde un programa Python?

Manejo de archivos

- Existen funciones predefinidas.
- Si las operaciones fallan, se levanta una **excepción**.
- Los archivos se manejan como objetos que se crean usando la función **open**.



La función open

```
f = open('archivo.txt', 'w')
```

- ¿De qué modo se abre este archivo? ¿Qué significa?
- Luego de la instrucción, ¿dónde se encuentra archivo.txt?
- ¿Cuándo puede dar un error esta sentencia?
- ¿Y la siguiente?

```
f = open('archivo.txt', 'x')
```

La función open

```
f = open('archivo.txt')
```

- ¿De qué modo se abre este archivo?

- ¿Cuándo puede dar un error esta sentencia?

La función open

- En realidad … hay más argumentos para open ...

```
open(file, mode='r', buffering=-1, encoding=None, errors=None,
     newline=None, closefd=True, opener=None)
```

- **encoding**: sólo para modo texto. Por defecto, la codificación establecida en las configuraciones del sistema.
- **errors**: sólo en modo texto. Es una cadena que dice qué hacer ante un error en la codificación/decodificación. ("strict", "ignore", ..)
- **newline**: sólo modo texto. Puede ser: None, '', 'n', 'r', y 'rn'.

```
f=open("/home/clau/pp.xxx", "r+", encoding="UTF-8")
```

Grabando datos en un archivo

- El caso más sencillo: guardanto texto en un archivo.

```
f = open('archivo.txt', 'w')
f.write('Hola, ')
f.write('Mundo!')
f.close()
```

- **write(cadena):** escribe *cadena* en el archivo y retorna cantidad de caracteres escritos.
- **close():** cierra el archivo.

Leyendo caracteres desde un archivo

```
f = open('archivo.txt', 'r')
print(f.read(4))
print(f.read())
```

- **read(cantidad_bytes):** lee *cantidad_bytes* del archivo.
- Si cantidad_bytes es <0 o no está, lee hasta fin de archivo.
- Retorna "" si EOF.
- Veamos algunos ejemplos que muestran otras formas de leer caracteres desde un archivo de texto.

¿Qué pasa si necesito guardar información que tiene una estructura?

- Ejemplo:
    - Los puntajes cada vez que juego a un juego. Información tipo: nombre jugador, puntaje, fecha.
    - El banco de preguntas: tema, enunciado, respuesta correcta.
    - Los Python Plus de los estudiantes por turnos: turno, nombre, apellido, num_alumno, cantidad_puntos, etc.
- En estos casos podría usar un archivo de texto: ¿cómo se les ocurre?

Algunas posibilidades

```
'nombre: Juan - puntaje: 1200 - fecha: 01/01/2019'

...

'''
nombre--Juan
puntaje--1200
fecha--01/01/2019
'''

...

'Juan-1200-01/01/2019'

...
'juan*1200*01/01/2019*'
```

- ¿Pros y contras?

Hay otras formas mejores...

JSON (JavaScript Object Notation)

- Es un formato de intercambio de datos muy popular.
- Ejemplo:

```
{
        "nombre": "Juan",
        "puntaje": "1200",
        "fecha": "01/01/2019"
}
```

- Veamos este ejemplo: https://developers.mercadolibre.com.ar/es_ar/categorias-y-publicaciones#close
- ¿Conocen la central meteorológica de la facultad? Miremos estos datos

---

JSON y Python

- Genero el archivo:

```
import json
import time

archivo = open("superHeroesEnJSON.txt", "w")
datos = [
{'nombre': 'Tony Stark', 'fecha': time.strftime("%a, %d %b %Y %H:%M:%S +0000", time.gmti
{'nombre': 'Bruce Wayne', 'fecha': time.strftime("%a, %d %b %Y %H:%M:%S +0000", time.gmt
json.dump(datos, archivo)
archivo.close()
```

- Lo leo:

```
import json
archivo = open("superHeroesEnJSON.txt", "r")
datos = json.load(archivo)
print(json.dumps(datos, sort_keys=True, indent=4))
archivo.close()
```

JSON y Python

- Se debe importar el módulo **json**.
- Permite serializar objetos.
- **dumps()** y **dump()**.
- **loads()** y **load()**.
- Más info en: https://docs.python.org/3/library/json.html

El módulo pickle

- El formato de datos que utiliza pickle es **específico de Python**.
- Se usan las funciones **load()** y **dump()**.
- También existen **loads()** y **dumps()**.
- Más info en: https://docs.python.org/3/library/pickle.html

El módulo pickle

- Genero el archivo:

```
import pickle
import time

archivo = open("superHeroesEnPICKLE.txt", "wb")
datos = [
{'nombre': 'Tony Stark', 'fecha': time.strftime("%a, %d %b %Y %H:%M:%S +0000", time.gmti
{'nombre': 'Bruce Wayne', 'fecha': time.strftime("%a, %d %b %Y %H:%M:%S +0000", time.gmt
pickle.dump(datos, archivo)
archivo.close()
```

- Lo leo:

```
import pickle
archivo = open("superHeroesEnPICKLE.txt", "rb")
datos = pickle.load(archivo)

for item in datos:
        for item, valor in item.items():
                print("{}: {}".format(item, valor))
archivo.close()
```

---

json vs. pickle

- ¿Qué consideraciones deberíamos analizar en cada caso?
- ¿Ventajas y desventajas?
- Si queremos intereactuar con aplicaciones externas, ¿qué formato les parece mejor?

---

json vs. pickle

- Pickle:
    - Archivos binarios.
    - Formato específico de Python. ¿Qué problema podría traer esto?

    - Tienen una advertencia de seguridad. ¿Por qué les parece?
- JSON:
    - Notación muy popular.
    - No dependemos de Python.
    - Archivos de texto.

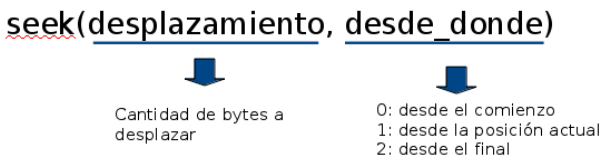
Pensemos en las siguientes situaciones

¿Qué hacemos si queremos:

- agregar nuevos datos a los datos ya existentes?;
- modificar el puntaje de un jugador específico en un juego?;
- agregar una nueva pregunta la banco de preguntas del juego de repaso?;
- incrementar los Python Plus de un estudiante determinado?

¿Qué tienen todas estas situaciones en común?

Acceso aleatorio



- Si el valor **desde_donde** no está, se asume 0.
- Si el archivo es de texto, solamente se considera desplazar desde el comienzo del archivo, es decir el valor **desde_donde** es 0.

Acceso aleatorio

```
archi = open("lineas2018.txt","w")
archi.write("abcdefghijk")
archi.close()

#PROBAR ABRIENDO ARCHIVO MODO TEXTO

archi = open("lineas2018.txt","rb")
archi.seek(2, 0)
print(archi.read(4).decode('ASCII'))
archi.seek(-1, 2) #Esto da error si el archivo se abre en modo"r"
print(archi.read().decode('ASCII'))
archi.close()
```

- Probemos esto mismo abriendo trabajando con archivos de texto.
- ¿archi.read(4).decode('ASCII')?

Acceso aleatorio

- **tell()**: retorna la posición actual.

```
archi = open("lineas2018.txt","rb")
archi.seek(0,2)
print(archi.tell())
archi.close()
```

CSV: ¿más formatos?

- CSV (Comma Separated Values).
- Es un formato común para importar/exportar desde/hacia hojas de cálculo y bases de datos.
- Funciones **reader()** y **writer()**.
- Veamos este ejemplo de uso del modulo csv
- Mujeres programadoras: http://mujeresprogramadoras.com.ar/
- ¿Vemos las egresadas de la UNLP?: mujeresProgramadoras.py

Entrada y salida estándar

- Tres archivos estándares: **stdin**, **stdout**, **stderr**.
- Usados por el intérprete como entrada, salida y error estándar.
- Se los accede a través del módulo **sys**.

```
import sys

sys.stdout.write("Hola \n\n que tal")
```

Entrada y salida estándar

```
nombre = input("Ingresa tu nombre")
```

Es equivalente a:

```
import sys

sys.stdout.write("Ingresa tu nombre")
nombre = sys.stdin.readline()
sys.stdout.write(nombre)
```

Tarea por Python Plus

- Agregar al programa juegos.py una función que guarde los datos del jugador y a qué juego jugó.
    - Primero: definamos la estructura de datos a utilizar.
    - Elijamos el formato de archivo a utilizar.
- ¿Lo implementan para sumar Python Plus?
- Tarea disponible hasta el **jueves 9 de mayo** a las 7:59.
- Si pasan el menú a PySimpleGUI, suman doble.

Retomamos los módulos estándares

Módulo OS

- Funciones para:
    - Procesamiento de archivos.
    - Directorios.
    - Permisos.
- Ejemplo:

```
import os
os.rename('lineas2019.txt','COPIA.TXT')
os.remove('COPIA.TXT')
```

Módulo OS

- Directorios: listdir()- mkdir() -chdir() - getcwd() - rmdir()

```
import os
lista = os.listdir("/home/clau/git/")
print(os.sep)
```

- Permisos: chmod()- access()

```
import os
print(os.access("/home/clau/git/", os.W_OK))
```

- Más info: http://docs.python.org/library/os.html#module-os

Módulo os.path

- Algunas funciones útiles: exists(), isdir(), isfile()

```
import os.path

print(os.path.exists("/home/clau/git/"))
print(os.path.isdir("/home/clau/git/"))
```

¿Qué vimos hoy?

Resumen

- Manejo básico de archivos.
- Formatos de impresión.
- Módulos csv, json y pickle.
- Más funciones de módulos estándares.