

Programación como
Herramienta para la
Ingeniería IIC2115

AYUDANTÍA 1

DICCIONARIOS

Los diccionarios corresponden a estructuras de datos orientadas a la asociación de pares de elementos mediante una relación: **llave-valor**. Se pueden crear de la siguiente manera:

```
nombre_diccionario = { llave_1 : valor_1, llave_2: valor_2 }
```

Para acceder a un valor se debe hacer lo siguiente:

```
nombre_diccionario[nombre_llave]
```

Los diccionarios poseen métodos útiles tales como:

- `keys()`: Devuelve una lista con todas las llaves del diccionario
- `values()`: Devuelve una lista con todos los valores del diccionario
- `Items()`: Devuelve una lista de tuplas de la forma (llave, valor)

Un punto importante para recordar es que las llaves de los diccionarios deben ser objetos **inmutables**

DICCIONARIOS

```
telefonos = {23545344: 'Juanito', 23545340: 'Andrea', 23545342: 'Ignacio'}
```

```
telefonos[23545344] # Se accede al valor 'Juanito'
```

```
telefonos.keys() # Devuelve [23545344, 23545340, 23545342]
```

```
telefonos.values() # Devuelve ['Juanito', 'Andrea', 'Ignacio']
```

```
telefonos.items() # Devuelve [(23545344, 'Juanito'), (23545340, 'Andrea'),  
(23545342, 'Ignacio')]
```

```
23545344 in telefonos # Devuelve True
```

```
12345678 in telefonos # Devuelve False
```

EJERCICIO 2 FORMATIVO 2 CAPÍTULO 1

En este ejercicio se nos entregaba las siguientes misiones:

- 1) Modelación de Entidades: Modelar al menos 3 entidades y las relaciones entre ellas en base a la información del archivo movies.json
- 2) Carga de Datos: Crear objetos de las entidades modeladas a partir de la información del archivo movies.json
- 3) Consultas sobre los datos:
 1. Encuentre los 5 géneros más populares
 2. Encuentre los 3 años con más películas estrenadas
 3. Encuentre a los 5 actores con la trayectoria más larga, es decir, mayor cantidad de años actuando
 4. Encuentre el reparto de una película (2 o más actores) que se haya repetido completo en otras la mayor cantidad de veces

ANALIZANDO ARCHIVO JSON

```
[{"title": "Catch My Smoke", "year": 1922, "cast": ["Tom Mix", "Lillian Rich", "Claude Payton"], "genres": ["Western"]}, {"title": "Caught Bluffing", "year": 1922, "cast": ["Frank Mayo", "Edna Murphy"], "genres": ["Western"]}, {"title": "Channing of the Northwest", "year": 1922, "cast": ["Eugene O'Brien", "Norma Shearer"], "genres": ["Drama"]}
...]
```

En base a la información que se puede observar del archivo json se puede decir que una película posee un título, un año de estreno, una lista de actores que participaron en la película y una lista de los géneros en los cuáles puede ser clasificada.

CARGANDO ARCHIVO MOVIES.JSON

```
import json  
  
with open("movies.json", encoding="utf8") as movies_file:  
    movies = json.load(movies_file)
```

{JSON}



python

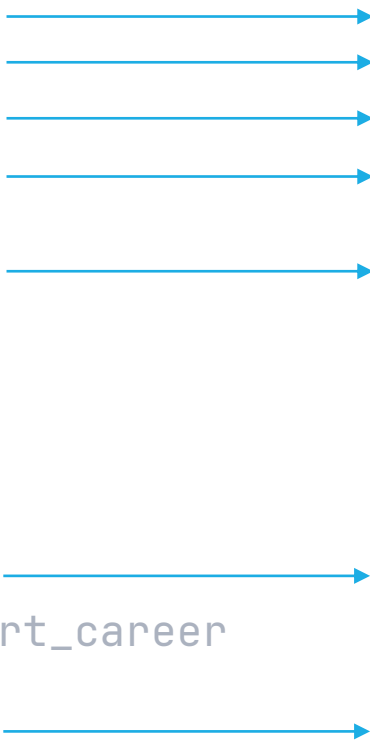
VARIABLE “MOVIES”

```
movies = [{"title": "Catch My Smoke", "year": 1922, "cast": ["Tom Mix", "Lillian  
Rich", "Claude Payton"], "genres": ["Western"]}, {"title": "Caught  
Bluffing", "year": 1922, "cast": ["Frank Mayo", "Edna  
Murphy"], "genres": ["Western"]}, {"ti": ["Eugene O'Brientle": "Channing of the  
Northwest", "year": 1922, "cast", "Norma Shearer"], "genres": ["Drama"]}]
```

La variable “movies” es una lista de diccionarios, esto significa que:

- `movies[0] = {"title": "Catch My Smoke", "year": 1922, "cast": ["Tom Mix", "Lillian Rich", "Claude Payton"], "genres": ["Western"]}`

MODELACIÓN DE ENTIDADES

<pre>class Actor: def __init__(self, full_name, start_career): self.full_name = full_name self.n_movies = 0 self.start_career = start_career self.end_career = start_career def update_track_record(self, year): if self.start_career > year: self.start_career = year if self.end_career < year: self.end_career = year def years_track_record(self): return self.end_career - self.start_career def __repr__(self): return f"{self.full_name}"</pre>		<p>Nombre completo del actor</p> <p>Número de películas en las que participa</p> <p>Año en el que empezó su carrera</p> <p>Año en el que terminó su carrera</p> <p>Método que actualiza el año en el que empezó su carrera o termino su carrera</p> <p>Método que retorna la cantidad de años que actuó</p> <p>Método que define que se mostrará al imprimir un objeto de Actor</p>
---	--	---

MODELACIÓN DE ENTIDADES

```
class Genre:  
    def __init__(self, name):  
        self.name = name  
        self.n_movies = 0
```

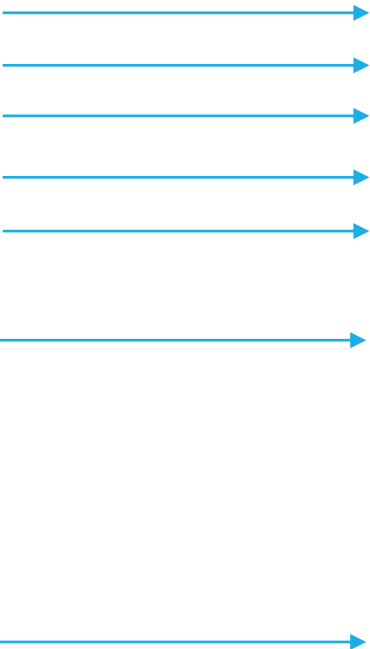
```
    def __repr__(self):  
        return f"{self.name}"
```

Nombre del género

Número de películas que pertenecen a este género

Método que define que se mostrará al imprimir un objeto de la clase Genre

MODELACIÓN DE ENTIDADES

<pre>class Movie: def __init__(self, title, year, cast, genres): self.title = title self.year = year self.cast = cast self.genres = genres self.add_info() def add_info(self): for actor in self.cast: actor.n_movies += 1 for genre in self.genres: genre.n_movies += 1 def __repr__(self): return f"{self.title} ({self.year}) - {self.cast} - {self.genres}"</pre>		<p>Título de la película</p> <p>Año de estreno de la película</p> <p>Elenco de la película, tupla de objetos Actor</p> <p>Géneros de la película, lista de objetos Genre</p> <p>Llamada al método add_info</p> <p>Método que actualiza la cantidad de películas en las que participo cada actor del elenco y el número de películas que tiene cada género al que pertenece esta película</p> <p>Método que define que se mostrará al imprimir un objeto de la clase Movie</p>
---	--	---

EJERCICIO 2 FORMATIVO 2 CAPÍTULO 1

En este ejercicio se nos entregaba las siguientes misiones:

- 1) **Modelación de Entidades: Modelar al menos 3 entidades y las relaciones entre ellas en base a la información del archivo movies.json**
- 2) **Carga de Datos: Crear objetos de las entidades modeladas a partir de la información del archivo movies.json**
- 3) **Consultas sobre los datos:**
 1. Encuentre los 5 géneros más populares
 2. Encuentre los 3 años con más películas estrenadas
 3. Encuentre a los 5 actores con la trayectoria más larga, es decir, mayor cantidad de años actuando
 4. Encuentre el reparto de una película (2 o más actores) que se haya repetido completo en otras la mayor cantidad de veces

CARGANDO DATOS

Carga de Datos: Crear objetos de las entidades modeladas a partir de la información del archivo movies.json. Se debe recordar que el atributo “cast” de un objeto de la clase Movie es una tupla que contiene objetos de la clase Actor y el atributo “genres” es una lista que contiene objetos de la clase Genre. También que el nombre de un actor puede estar en el “cast” de distintas películas y lo mismo para un género. Lo que se esperaría lograr sería que los objetos de la clase Movie sean de la siguiente manera:

```
movie_1 = Movie("Spiderman 1", 2002, (actor_1, actor_2, ...), [genero_1, genero_2, ...])
movie_2 = Movie("Spiderman 2", 2004, (actor_1, actor_2, ...), [genero_1, genero_2, ...])
```

En los que actor_1, un objeto de la clase Actor, presente en ambos objetos de la clase Movie sean iguales, sin embargo, se debe tener en cuenta lo siguiente:

```
tobey_maguire_1 = Actor("Tobey Maguire", 1987)
tobey_maguire_2 = Actor("Tobey Maguire", 1987)
print(tobey_maguire_1 == tobey_maguire_2 ) # False
```

¿ Cómo lograr lo que sean igual y usar el mismo objeto para cada caso?

CARGANDO DATOS

Respuesta: Usando diccionarios

Lo que se hará para cada objeto de cada clase será almacenarlos en diccionarios, los cuales serán de la siguiente manera:

- Diccionario para actores: Este diccionario tendrá como llave el nombre del actor(string) y como valor un objeto de la clase Actor cuyo nombre será el mismo que el de la llave.
- Diccionario para géneros: Este diccionario tendrá como llave el nombre del género(string) y como valor un objeto de la clase Genre cuyo nombre será el mismo que el de la llave.
- Diccionario para películas: Este diccionario tendrá como llave una tupla que será de la forma: (nombre_película, año_película, tupla_objetos_actor) y como valor un objeto de la clase Movie cuyo nombre, año y cast serán los mismo que los de su llave. Se usa esta llave ya que en el archivo json se encontraron películas que coincidían con nombre y año, pero se diferenciaban en el elenco que usaban.

CARGANDO DATOS

```
actors_dict = {}  
genres_dict = {}  
movies_dict = {}
```

→ Se crean diccionarios para actores, géneros y películas

```
for movie in movies:  
    cast = []  
    genres = []
```

→ Se recorre la lista "movies" y por cada iteración se crean las listas que almacenaran los objetos de Actor y Genre para el objeto de la clase Movie que se creará al final

```
for actor in movie["cast"]:  
    if actor not in actors_dict:  
        actors_dict[actor] = Actor(actor, movie["year"])  
    else:  
        actors_dict[actor].update_track_record(movie["year"])  
    cast.append(actors_dict[actor])
```

→ Se recorre la lista de actores de la película. Por cada valor en la lista se revisa si en "actors_dict" hay una llave con el nombre del actor, en caso de no estar en el diccionario se crea un objeto Actor y se guarda usando como la llave el nombre del actor, caso contrario se llama al método update_track_record. Al final se añade a la lista "cast" el objeto Actor obteniéndolo desde el diccionario.

```
for genre in movie["genres"]:  
    if genre not in genres_dict:  
        genres_dict[genre] = Genre(genre)  
    genres.append(genres_dict[genre])
```

→ Se recorre la lista de géneros de la película. Por cada valor en la lista se revisa si en "genres_dict" hay una llave con el nombre del género, en caso de no estar en el diccionario se crea un objeto Genre y se guarda usando como la llave el nombre del género. Al final se añade a la lista "genres" el objeto Genre obteniéndolo desde el diccionario.

```
if (movie["title"], movie["year"], tuple(cast)) not in movies_dict:  
    movies_dict[(movie["title"], movie["year"], tuple(cast))] =  
    Movie(movie["title"], movie["year"], tuple(cast), genres)  
  
else:  
    print("*"*80)  
    print(f"Película duplicada: {movie['title']} - ({movie['year']}) -  
{tuple(cast)} - {genres}")  
    movie_found = movies_dict[(movie["title"], movie["year"], tuple(cast))]  
    print(f"En el diccionario ya está: {movie_found}")  
    print("*"*80)
```

→ Se revisa si la película está presente en el diccionario, en caso de no estarlo se crea un objeto Movie y se guarda en el diccionario "movies_dict" bajo la llave "(title, year, cast)"

CARGANDO DATOS

```
actors_dict = {}  
genres_dict = {}  
movies_dict = {}
```

```
cast = []  
genres = []  
  
for actor in movie["cast"]:  
    if actor not in actors_dict:  
        actors_dict[actor] = Actor(actor, movie["year"])  
    else:  
        actors_dict[actor].update_track_record(movie["year"])  
    cast.append(actors_dict[actor])
```

Se recorre la lista de actores de la película, tener en cuenta que “actor” es un string con el nombre del actor. Por cada valor en la lista se revisa si en “actors_dict” hay una llave con el nombre del actor, en caso de no estar en el diccionario se crea un objeto Actor y se guarda usando como la llave del nombre del actor, caso contrario se llama al método update_track_record. Al final se añade a la lista “cast” el objeto Actor obteniéndolo desde el diccionario.

CARGANDO DATOS

```
actors_dict = {}  
genres_dict = {}  
movies_dict = {}
```

```
cast = []  
genres = []  
  
for genre in movie["genres"]:  
    if genre not in genres_dict:  
        genres_dict[genre] = Genre(genre)  
    genres.append(genres_dict[genre])
```

Se recorre la lista de géneros de la película , tener en cuenta que “genre” es un string con el nombre del género. Por cada valor en la lista se revisa si en “genres_dict” hay una llave con el nombre del género, en caso de no estar en el diccionario se crea un objeto Genre y se guarda usando como la llave el nombre del género Al final se añade a la lista “genres” el objeto Genre obteniéndolo desde el diccionario.

CARGANDO DATOS

```
actors_dict = {}  
genres_dict = {}  
movies_dict = {}
```

```
        if (movie["title"],  
movie["year"], tuple(cast)) not in movies_dict:  
            movies_dict[(movie["title"],  
movie["year"], tuple(cast))] =  
            Movie(movie["title"], movie["year"], tuple(cast),  
genres)  
  
        else:  
            print("*"*80)  
            print(f"Película duplicada:  
{movie['title']} - ({movie['year']}) -  
{tuple(cast)} - {genres}")  
            movie_found = movies_dict[(movie["title"],  
movie["year"], tuple(cast))]  
            print(f"En el diccionario ya está:  
{movie_found}")  
            print("*"*80)
```

Se revisa si la película está presente en el diccionario, en caso de no estarlo se crea un objeto Movie y se guarda en el diccionario "movies_dict" bajo la llave "(title, year, cast)"

EJERCICIO 2 FORMATIVO 2 CAPÍTULO 1

En este ejercicio se nos entregaba las siguientes misiones:

- 1) **Modelación de Entidades:** Modelar al menos 3 entidades y las relaciones entre ellas en base a la información del archivo `movies.json`
- 2) **Carga de Datos:** Crear objetos de las entidades modeladas a partir de la información del archivo `movies.json`
- 3) Consultas sobre los datos:
 1. Encuentre los 5 géneros más populares
 2. Encuentre los 3 años con más películas estrenadas
 3. Encuentre a los 5 actores con la trayectoria más larga, es decir, mayor cantidad de años actuando
 4. Encuentre el reparto de una película (2 o más actores) que se haya repetido completo en otras la mayor cantidad de veces

FUNCIONES LAMBDA

Son funciones pequeñas y sencillas en una sola línea de código. Son funciones que no necesitan ser reutilizadas en otras partes del código

`lambda argumentos: valor a regresar`

Ejemplo:

```
square = lambda x: x**2
```

FUNCIÓN SORTED

Es una función de Python que, a partir de un iterable, cualquier objeto que se puede recorrer, retorna una lista con el iterable ordenado bajo algún criterio, tiene la siguiente sintaxis:

```
sorted(iterable, key=None, reverse=False)
```

- **iterable**: objeto que se quiere ordenar, el cual es posible recorrer por ejemplo una lista
- **key**: función a ejecutar para decidir el orden
- **reverse**: un booleano para decidir si se ordena de forma ascendente (False) o descendente (True)

FUNCIÓN MAX

Es una función de Python que se utiliza para obtener el mayor valor en un iterable

```
max(iterable, key=None, default=None)
```

- `iterable`: objeto del cual se quiera obtener su mayor valor, el cual es posible recorrer por ejemplo una lista
- `key`: función a ejecutar que decide el criterio bajo el cual se obtiene el valor máximo
- `default`: valor opcional que se devuelve si el iterable está vacío

CONSULTAS

1. Encuentre los 5 géneros más populares

```
def popular_genres(genres_dict):  
    sorted_list_genres =  
sorted(genres_dict.values(), key=lambda x:  
x.n_movies, reverse=True)  
    most_popular_genres =  
sorted_list_genres[:5]  
    print("Los 5 géneros más populares son:")  
    for genre in most_popular_genres:  
        print(f"{genre.name} hay  
{genre.n_movies} películas de este género")
```

En primer lugar, “genres_dict.values()” retorna una lista con los valores del diccionario “genres_dict”, es decir, sería algo así:

```
genres_dict.values() = [ObjetoGenres_1, ObjetoGenres_2, ObjetoGenres_3]  
<<ObjetoGenres_1.n_movies = 2>>  
<<ObjetoGenres_2.n_movies = 45>>  
<<ObjetoGenres_3.n_movies = 4>>
```

Se usa la función “sorted” para ordenar en forma descendente esta lista de acuerdo con el atributo “n_movies” de cada objeto, esto se logra al usar la función lambda al parámetro “key”:

```
lambda x: x.n_movies
```

Esta función lambda dice que retorna el atributo n_movies del parámetro “x”, y “x” tomará cada elemento de la lista.

Se toma los 5 primeros elementos de la lista ordenada y se guarda en la variable “most_popular_genres”:

```
most_popular_genres = [ObjetoGenres_2, ObjetoGenres_3, ObjetoGenres_1]
```

Finalmente, se imprime los valores de esta última lista con un mensaje adecuado.

CONSULTAS

2. Encuentre los 3 años con más películas estrenadas

```
def years_most_premiers(movies_dict):
    premiers_dict = {}
    for movie in movies_dict.values():
        if movie.year not in premiers_dict:
            premiers_dict[movie.year] = 0
        premiers_dict[movie.year] += 1
    sorted_list_years =
sorted(premiers_dict.items(), key=lambda x:
x[1], reverse=True)
    list_years = sorted_list_years[:3]
    print("Los 3 años con más películas
estrenadas son:")
    for data in list_years:
        print(f"El año {data[0]} con {data[1]}
películas producidas")
```

En primer lugar, se creará un diccionario llamado "premiers_dict" su contenido será de la siguiente manera:

premiers_dict = { año_1: número de películas estrenadas en el año_1, ..}

Se recorrerá la lista de objetos Movies, que entrega "movies_dict_values()", se revisará si el año de estreno de la película, entregado por el atributo "year" de cada objeto Movie, este en las llaves que posee "premiers_dict". En caso de no estar se guarda el valor 0 bajo la llave "movie.year", esto es así: premiers_dict = { 1952: 0 } y posteriormente se aumenta en 1 para contabilizar el encuentro actual. En caso de ya estar presente la llave solo se aumenta el valor en 1.

Teniendo en cuenta que "premiers_dict.items()" entre una lista de tuplas de la forma (llave, valor) del diccionario premiers_dict, se crea una lista ordenada con la función "sorted", se ordena en base al segundo elemento de cada tupla, esto se logra al usar la función lambda al parámetro "key": lambda x: x[1], el valor de "x" sería cada una tupla y se dice que se ordené según el segundo valor de cada tupla en forma descendente. Luego se crea una lista con los 3 primeros elementos, se recorre la lista y se imprime un mensaje adecuado

CONSULTAS

3. Encuentre a los 5 actores con la trayectoria más larga, es decir, mayor cantidad de años actuando

```
def longer_track_record(actors_dict):  
    sorted_list_actors = sorted(actors_dict.values(),  
key=lambda x: x.years_track_record(), reverse=True)  
    list_actors = sorted_list_actors[:5]  
    print("Los 5 actores con trayectoria más larga  
son:")  
    for actor in list_actors:  
        print(f"{actor.full_name} con  
{actor.years_track_record()} años de trayectoria")
```

En primer lugar, “actors_dict.values()” retorna una lista con los valores del diccionario “actors_dict”, es decir, sería algo así:
genres_dict.values() = [ObjetoActor_1, ObjetoActor_2, ObjetoActor_3]

Se usa la función “sorted” para ordenar en forma descendente esta lista de acuerdo con el resultado del método “years_track_record()” de cada objeto, esto se logra al usar la función lambda al parámetro “key”:

lambda x: x.years_track_record()

Esta función lambda dice que retorna el valor de ejecutar el método “years_track_record” del parámetro “x”, y “x” tomará cada elemento de la lista.

Se toma los 5 primeros elementos de la lista ordenada y se guarda en la variable “list_actors”. Finalmente, se imprime los valores de esta última lista con un mensaje adecuado.

CONSULTAS

4. Encuentre el reparto de una película (2 o más actores) que se haya repetido completo en otras la mayor cantidad de veces

```
def most_repeated_casts(movies_dict):  
    cast_dict = {}  
    for movie in movies_dict.values():  
        if len(movie.cast) >= 2:  
            if movie.cast not in cast_dict:  
                cast_dict[movie.cast] = 0  
            cast_dict[movie.cast] += 1  
        else:  
            continue  
    max_value = max(cast_dict.items(),  
key=lambda x: x[1])  
    print(f"El reparto de una película que  
más se ha repetido es {max_value[0]} con  
{max_value[1]} repeticiones")
```

En primer lugar, se creará un diccionario llamado "cast_dict" su contenido será de la siguiente manera:

cast_dict = { (ObjetoActor_1, ObjetoActor_2, ...) : número de veces que apareció este elenco }

Se recorrerá la lista de objetos Movies, que entrega

"movies_dict.values()", se revisará si el elenco de la película, entregado por el atributo "cast" de cada objeto Movie, este en las llaves que posee "cast_dict". En caso de no estar se guarda el valor 0 bajo la llave

"movie.cast", esto es así: cast_dict = {(ObjetoActor_1, ObjetoActor_2, ...) : 0 } y posteriormente se aumenta en 1 para contabilizar el encuentro actual. En caso de ya estar presente la llave solo se aumenta el valor en 1.

Teniendo en cuenta que "cast_dict.items()" entre una lista de tuplas de la forma (llave, valor) del diccionario cast_dict, se obtiene el valor máximo de esta lista según la cantidad de veces que apareció el elenco, esto se logra al usar la función max indicando que su "key" es la función lambda:

lambda x: x[1], el valor de "x" sería cada una tupla y se dice que se ordenó según el segundo valor de cada tupla en forma descendente. Se recorre la lista y se imprime un mensaje adecuado

EJERCICIO 2 FORMATIVO 2 CAPÍTULO 1

En este ejercicio se nos entregaba las siguientes misiones:

- 1) Modelación de Entidades: Modelar al menos 3 entidades y las relaciones entre ellas en base a la información del archivo movies.json
- 2) Carga de Datos: Crear objetos de las entidades modeladas a partir de la información del archivo movies.json
- 3) Consultas sobre los datos:
 1. Encuentre los 5 géneros más populares
 2. Encuentre los 3 años con más películas estrenadas
 3. Encuentre a los 5 actores con la trayectoria más larga, es decir, mayor cantidad de años actuando
 4. Encuentre el reparto de una película (2 o más actores) que se haya repetido completo en otras la mayor cantidad de veces