

Guía de ejercicios - Efemérides y Mini Presentador



¡Hola! Te damos la bienvenida a esta nueva guía de estudio.

¿En qué consiste esta guía?

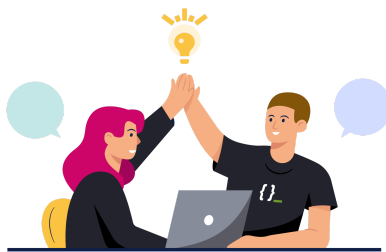
La siguiente guía de estudio tiene como objetivo practicar y ejercitar los contenidos que hemos visto en clase.

¡Vamos con todo!



Tabla de contenidos

| | |
|------------------------------------|---|
| Actividad guiada: Mini presentador | 2 |
| Actividad guiada: Efemérides | 5 |



¡Comencemos!



Actividad guiada: Mini presentador

Aplicar cómo combinar la interpolación con `argv` para poder importar datos a Python. Sigamos los siguientes pasos:

1. Crear el archivo `argumentos.py`.
2. Lo primero que haremos en nuestro archivo, es importar el módulo que nos permite rescatar argumentos desde el terminal:

```
import sys
```

3. Dentro de este módulo, llamaremos a la propiedad `argv`, la cual corresponde a la lista de los argumentos introducidos en el terminal.

El primer argumento (cuando se ejecuta un script) **es el nombre del mismo**, y se encuentra ubicado en la posición **0** de la lista de argumentos.

Todo lo que se escriba después del nombre del script, corresponderá a los argumentos siguientes.

Crearemos una variable **nombre**, a la cual le asignaremos el argumento en la posición **1** de la lista de argumentos y la variable **apellido** con la posición **2**, de la siguiente forma:

```
nombre = sys.argv[1]  
apellido = sys.argv[2]
```

4. Ahora visualicemos los resultados ingresados agregando lo siguiente:

```
print(f"Mi nombre es {nombre}")  
print(f"Mi apellido es {apellido}")  
print(f" nombre de este archivo es {sys.argv[0]}")
```

El código completo se ve así:

```
argumentos.py X
argumentos.py > ...
1 import sys
2
3 nombre = sys.argv[1]
4 apellido = sys.argv[2]
5
6 print(f'Mi nombre es {nombre}')
7 print(f'Mi apellido es {apellido}')
8 print(f'El nombre de este archivo es { sys.argv[0] }')
```

Imagen 1. Código final de argumentos.py
Fuente: Desafío Latam

5. Este script se ejecutará de la misma manera que cualquier script de Python, pero será necesario agregar los argumentos solicitados de la siguiente forma:

```
~/Desktop/PYTHON
python argumentos.py Carlos Santana
Mi nombre es Carlos
Mi apellido es Santana
El nombre de este archivo es argumentos.py
```

Imagen 2. Ejecutando argumentos.py en terminal
Fuente: Desafío Latam

Como se puede observar, las palabras Carlos y Santana que acompañan la ejecución del programa, se utilizarán como los argumentos `argv` en posiciones 1 y 2 respectivamente. Por otro lado, el argumento 0 siempre corresponderá al nombre de archivo en ejecución.

Si ejecutamos `type(sys.argv)` notaremos que se trata de una lista, esa es la razón por la que podemos utilizar índices para acceder a distintas partes de este. Es importante recordar que todas las operaciones y métodos que veamos para las listas se aplicarán también para el `sys.argv`.

```
import sys
type(sys.argv)
```

```
list
```



NOTA: El inconveniente que tiene `sys.argv` es que es poco intuitivo para el usuario, ya que si no está familiarizado con el código, no podrá entender cada argumento inicialmente. Pero es fundamental cuando existen procesos automáticos agendados por el computador el cual ejecuta un archivo .py para realizar procesos claves para las empresas.

Descarga del LMS el archivo "Actividad guiada - Mini presentador.zip"



Actividad guiada: Efemérides

Toda fecha tiene sucesos importantes ¿cómo recordarlos? Las efemérides son conjunto de acontecimientos importantes ocurridos en una misma fecha, por lo que aquí intentaremos utilizar lo aprendido para almacenar eventos importantes y consultarlos.

Supongamos los siguientes eventos:

- 1 de Enero: Año Nuevo
- 27 de Febrero: Terremoto en Chile
- 8 de Marzo: Día de la mujer
- 21 de Mayo: Glorias Navales
- 18 de Septiembre: Fiestas Patrias
- 19 de Septiembre: Glorias del Ejército
- 25 de Diciembre: Navidad

1. Crear el archivo `efemerides.py`

2. Disponibilizar estos elementos en Python

Notamos que queremos almacenar varios elementos en Python, naturalmente esto nos indica que requerimos una estructura de datos, ¿pero cuál?

Cualquiera sirve para almacenar pero los requerimientos piden también consultarlos.

Las listas y las tuplas requieren índices por lo que no serían lo más apropiado, por lo que la opción que nos queda son los Diccionarios:

```
# definimos el diccionario
efemerides = {'1 de Enero': 'Año Nuevo',
              '27 de Febrero': 'Terremoto en Chile',
              '8 de Marzo': 'Día de la Mujer',
              '21 de Mayo': 'Glorias Navales',
              '18 de Septiembre': 'Fiestas Patrias',
              '19 de Septiembre': 'Glorias del Ejercito',
              '25 de Diciembre': 'Navidad'}
```

3. Consultar la Fecha

Mediante `input()` se puede solicitar al usuario que ingrese la fecha a consultar.

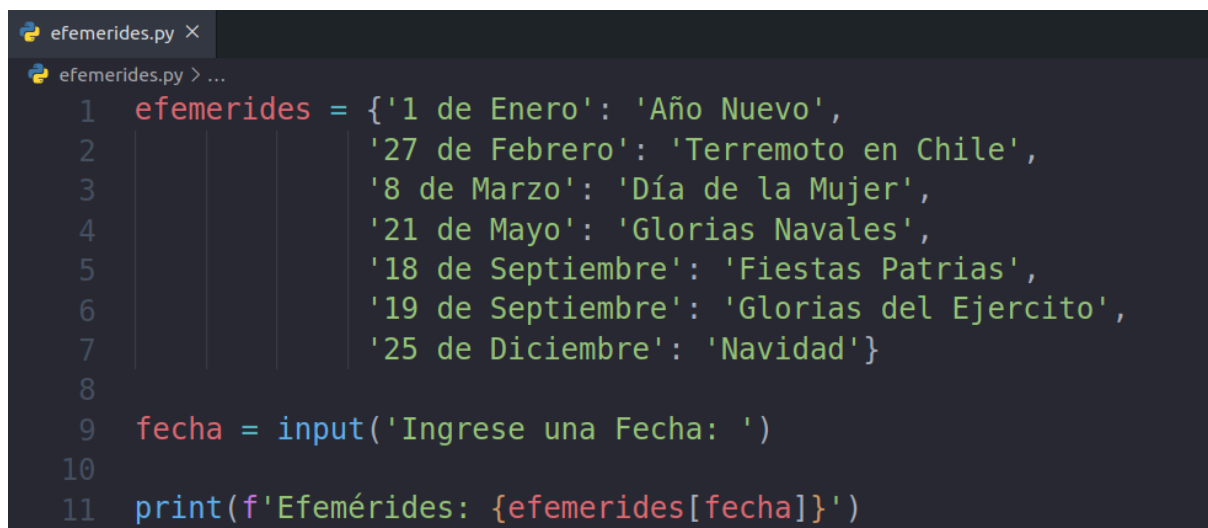
```
fecha = input('Ingrese una Fecha:')
```

4. Mostrar los resultados

Para mostrar los resultados es necesario identificar la clave del diccionario, que en este caso particular es la fecha. Combinando esto con f-strings el código queda así:

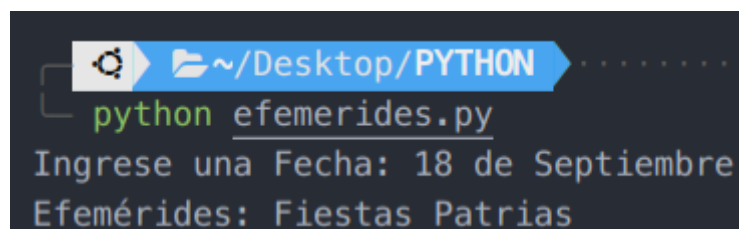
```
print(f'Efemerides: {efemerides[fecha]}')
```

El código completo se ve así:



```
efemerides.py x
efemerides.py > ...
1 efemerides = {'1 de Enero': 'Año Nuevo',
2               '27 de Febrero': 'Terremoto en Chile',
3               '8 de Marzo': 'Día de la Mujer',
4               '21 de Mayo': 'Glorias Navales',
5               '18 de Septiembre': 'Fiestas Patrias',
6               '19 de Septiembre': 'Glorias del Ejercito',
7               '25 de Diciembre': 'Navidad'}
8
9 fecha = input('Ingrese una Fecha: ')
10
11 print(f'Efemerides: {efemerides[fecha]}')
```

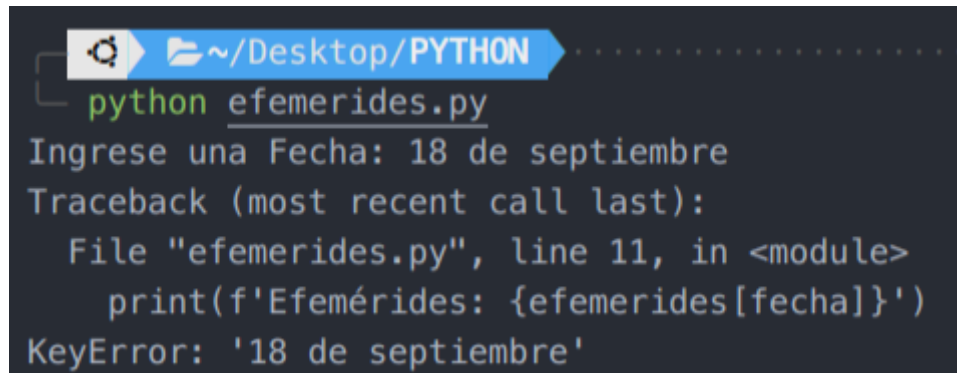
Imagen 3. Vista completa de efemerides.py
Fuente: Desafío Latam.



```
~/Desktop/PYTHON
python efemerides.py
Ingrese una Fecha: 18 de Septiembre
Efemerides: Fiestas Patrias
```

Imagen 4. Ejecución de efemerides.py
Fuente: Desafío Latam.

Pero vamos más allá. Si bien este programa funciona tiene algunos inconvenientes. Por ejemplo:

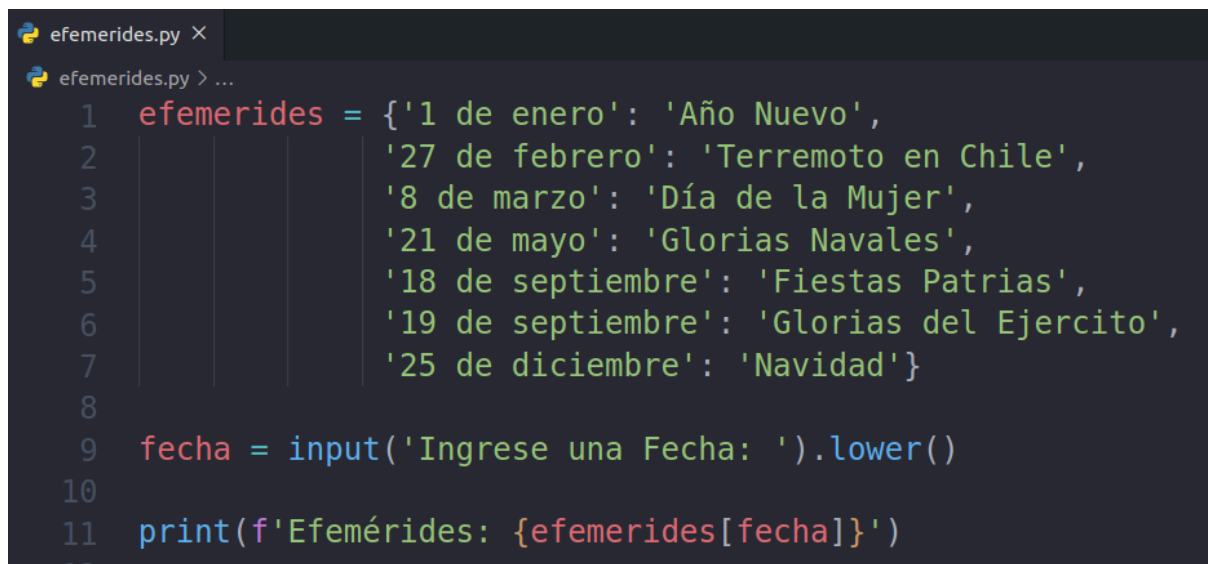


```
python efemerides.py
Ingrese una Fecha: 18 de septiembre
Traceback (most recent call last):
  File "efemerides.py", line 11, in <module>
    print(f'Efemérides: {efemerides[fecha]}')
KeyError: '18 de septiembre'
```

Imagen 5. Detectando algunos problemas en Vista completa de efemerides.py
Fuente: Desafío Latam.

Si ingresamos “18 de septiembre” (notar la minúscula), el programa falla. ¿Qué estrategia podemos aplicar para evitar este problema?

Podríamos definir todas las claves en minúsculas y aplicar `.lower()` a `input()` así:



```
efemerides.py X
efemerides.py > ...
1 efemerides = {'1 de enero': 'Año Nuevo',
2               '27 de febrero': 'Terremoto en Chile',
3               '8 de marzo': 'Día de la Mujer',
4               '21 de mayo': 'Glorias Navales',
5               '18 de septiembre': 'Fiestas Patrias',
6               '19 de septiembre': 'Glorias del Ejercito',
7               '25 de diciembre': 'Navidad'}
8
9 fecha = input('Ingrese una Fecha: ').lower()
10
11 print(f'Efemérides: {efemerides[fecha]}')
```

Imagen 6. Vista completa de efemerides.py
luego de solucionar el problema.
Fuente: Desafío Latam.



NOTA: Todas las fechas ahora están en minúsculas por lo que se soluciona el problema antes mostrado:

```
~/Desktop/PYTHON
python efemerides.py
Ingresa una Fecha: 18 de septiembre
Efemérides: Fiestas Patrias
```

Imagen 7. Ejecutando el programa luego de la corrección.

Fuente: Desafío Latam.

Ahora bien, vamos aún más allá. Siempre existe la posibilidad de que un usuario ingrese información no disponible en nuestro programa, como por ejemplo:

```
~/Desktop/PYTHON
python efemerides.py
Ingresa una Fecha: 5 de Febrero
Traceback (most recent call last):
  File "efemerides.py", line 11, in <module>
    print(f'Efemérides: {efemerides[fecha]}')
KeyError: '5 de febrero'
```

Imagen 8. Detectando oportunidades de mejora

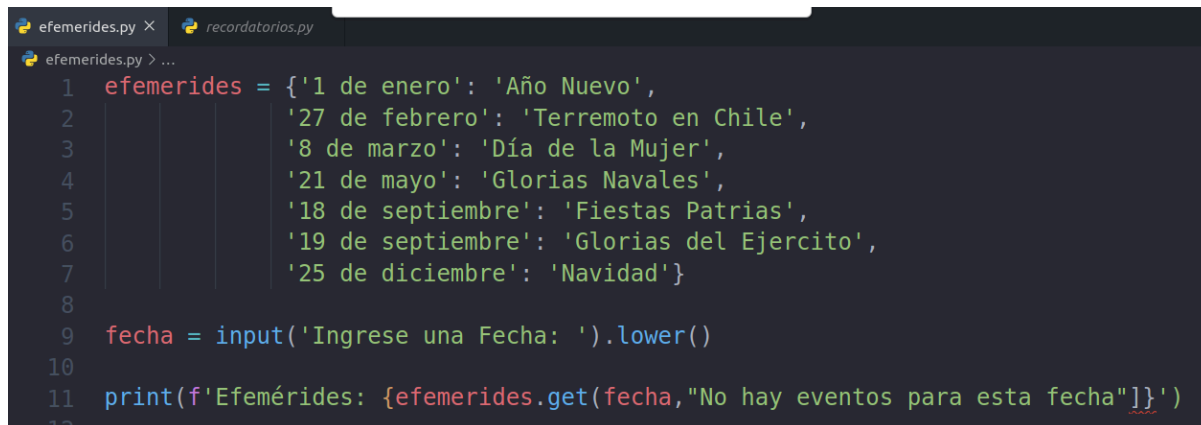
Fuente: Desafío Latam.

Al ingresar una fecha que no tiene información nuestro programa fallará, ya que no es capaz de encontrar la clave especificada, para evitar aquello podemos utilizar `.get()`. Lo bueno que tiene este método, es que permite añadir una salida alternativa en caso de no encontrar un elemento.

```
# en caso de no encontrar la fecha solicitada devolverá el texto
indicado
print(f'Efemérides: {efemerides.get(fecha,"No hay eventos para esta
fecha")})')
```

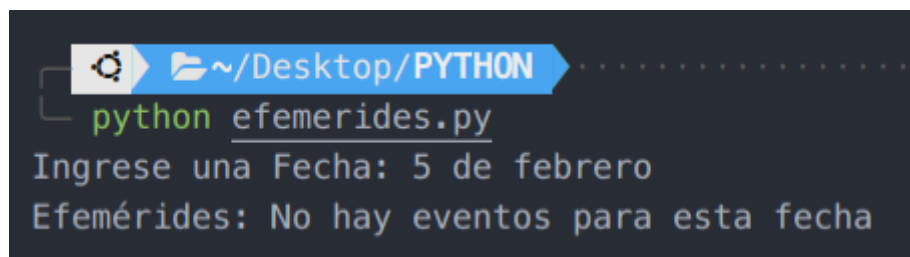



NOTA: Fijarse en el uso de los dos tipos de comillas. Esto, porque la comilla simple (') genera el **f-string** mostrado en pantalla por `print()`, mientras que la comilla doble (") genera el string alternativo en caso de no encontrar una clave. Considera que, de no respetar esto, Python arrojará error.



```
efemerides.py × recordatorios.py
efemerides.py > ...
1 efemerides = {'1 de enero': 'Año Nuevo',
2               '27 de febrero': 'Terremoto en Chile',
3               '8 de marzo': 'Día de la Mujer',
4               '21 de mayo': 'Glorias Navales',
5               '18 de septiembre': 'Fiestas Patrias',
6               '19 de septiembre': 'Glorias del Ejercito',
7               '25 de diciembre': 'Navidad'}
8
9 fecha = input('Ingrese una Fecha: ').lower()
10
11 print(f'Efemérides: {efemerides.get(fecha,"No hay eventos para esta fecha")}')
12
```

Imagen 9. Vista completa de efemerides.py
luego de agregar el método get().
Fuente: Desafío Latam.



```
~/Desktop/PYTHON
python efemerides.py
Ingrese una Fecha: 5 de febrero
Efemérides: No hay eventos para esta fecha
```

Imagen 10. Ejecutando el programa sin error.
Fuente: Desafío Latam.

Descarga del LMS el archivo "Actividad guiada - Efemérides"