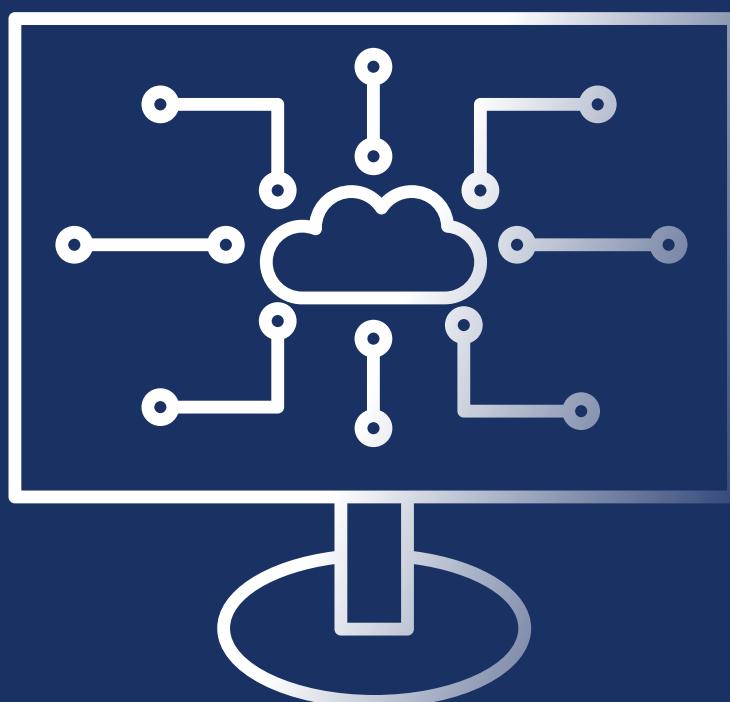


Programación I



Lic. Julia Monasterio
marmonasterio@uade.com.ar



Unidades

-  **Unidad 1:** Repaso de matrices, estructura, operaciones y uso en funciones
-  **Unidad 2:** Implementación y gestión en proyectos en Git
-  **Unidad 3:** Listas avanzadas, cadena de caracteres y expresiones regulares.
-  **Unidad 4:** Manipulación avanzada de diccionarios, tuplas y conjuntos.
-  **Unidad 5:** Excepciones y pruebas unitarias
-  **Unidad 6:** Archivos
-  **Unidad 7:** Recursividad

Clase N° 6

TEMAS

- Utilización de expresiones regulares en Python
- Diccionarios

Utilizacion de expresiones regulares en Python

- Python tiene un modulo especifico para trabajar con expresiones regulares

`import re`

Utilizacion de expresiones regulares en Python

- **Método match:** se utiliza para determinar si una cadena coincide con un patrón de expresión regular desde el inicio de la cadena.
- Retorna true si encuentra una coincidencia exitosa evaluándose en un contexto booleano (ejemplo if)
- Si no se evalúa en un contexto booleano retornara el objeto match en caso de coincidencia o None en caso de no coincidencia.

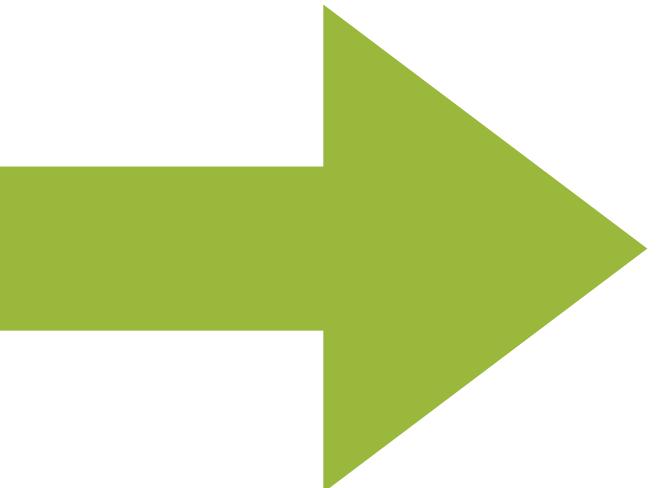
Utilizacion de expresiones regulares en Python

- **Método match:** el primer parámetro debe ser el patrón y luego la cadena en la que se desea buscar

Ejemplo Match

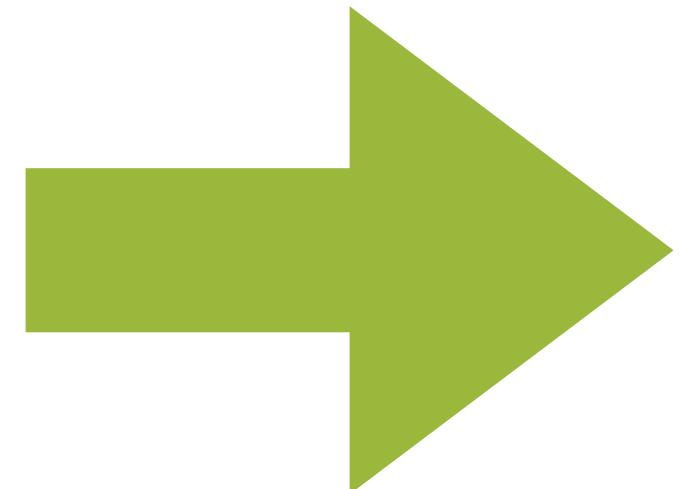
```
from re import match  
  
patron="\d{3}"  
cadena="123"  
  
resultado= match(patron,cadena)  
  
print(resultado)
```

```
match(patron, cadena)  
<re.Match object; span=(0, 3), match='123'>
```



Ejemplo Match

```
from re import match  
  
patron="\d{3}"  
cadena="d123"  
  
resultado= match(patron,cadena)  
  
print(resultado)
```



None

Ejemplo Match en if

```
from re import match
cadenas=["A123","B456","C789","D87764"]
patron="[A-Za-z][1-3]{3}"
for cadena in cadenas:
    if match(patron,cadena):
        print(f"La cadena {cadena} cumple con el patron")
    else:
        print(f"La cadena {cadena} no cumple con el patron")
```

La cadena A123 cumple con el patron
La cadena B456 no cumple con el patron
La cadena C789 no cumple con el patron
La cadena D87764 no cumple con el patron

Utilizacion de expresiones regulares en Python

- **Método search:** a diferencia de match que verifica solo el inicio de la cadena, search examina toda la cadena en busca de la primera ocurrencia del patrón especificado.
- Si encuentra coincidencia devuelve un objeto Match con varia informacion, si no encuentra devuelve None.

Utilizacion de expresiones regulares en Python

- **El objeto `match` contiene información sobre la coincidencia encontrada:**
 - **.group()** : retorna la parte de la cadena que coincidió con el patrón
 - **.start()** : retorna la posición inicial de la coincidencia
 - **.end()** : retorna la posición final de la coincidencia
 - **.span()** retorna una tupla con la posición de inicio y fin de coincidencia

Ejemplo Search

Iniciación I > Curso Beigano > Clase 8 > Ejemplo.py > ...

```
from re import match,search

cadena="El precio del producto es $11.33."
patron="[0-9]+"

match=search(patron,cadena)

if match:
    print(f"Primer coincidencia encontrada {match.group()}")
    print(f"La coincidencia empieza en el indice {match.start()}")
    print(f"La coincidencia termina en el indice {match.end()}")
    print(f"La coincidencia comienza y termina en {match.span()}")
else:
    print("No hay match")
```

```
Primer coincidencia encontrada 11
La coincidencia empieza en el indice 27
La coincidencia termina en el indice 29
La coincidencia comienza y termina en (27, 29)
```

Metodo search: omitir mayusculas y minusculas

- Para utilizar el método search y omitir las diferencias entre mayúsculas y minúsculas, se puede hacer uso de un tercer parámetro que es **re.IGNORECASE**
- **Esto hace que la búsqueda del patrón de expresión regular no sea sensible a mayúsculas o minúsculas**

Metodo search: omitir mayusculas y minusculas - Ejemplo

```
from re import match,search,IGNORECASE

cadena="La UADE es una Universidad Argentina"
patron="universidad"

match=search(patron,cadena,IGNORECASE)

if match:
    print(f"Se encontro {match.group()} en la posicion {match.start()}")
else:
    print("No hay match")
```

Utilizacion de expresiones regulares en Python

- **Método findall:** se utiliza para encontrar **todas las ocurrencias** de un patrón de expresión regular en una cadena de texto y devolverlas como una lista de cadenas.
- A diferencia de search que solo devuelve la primer ocurrencia
- Utiliza dos argumentos que es el patrón y la cadena

Ejemplo:

```
from re import match,search,IGNORECASE,findall

cadena="Juan tiene 34 años Pedro tiene 14 años Maria tiene 55 años"
patron="[0-9]{2}"

edades=findall(patron,cadena)

if edades:
    for edad in edades:
        print(edad)
else:
    print("No hay match")
```

Utilizacion de expresiones regulares en Python

Método sub: se utiliza para reemplazar todas las ocurrencias de un patrón de expresión regular en una cadena de texto con otra cadena especificada

Es útil cuando se desea modificar o limpiar texto según un patrón definido.

Recibe tres parámetros:

- patrón de expresion regular
- cadena que se utilizara para reemplazar las coincidencias encontradas
- cadena en la que se realizara la búsqueda y el reemplazo

Utilizacion de expresiones regulares en Python

```
from re import match,search,IGNORECASE,findall,sub

cadena="El numero de telefono de mi casa es 11-9999-4444"
patron="[0-9]{2}-[0-9]{4}-[0-9]{4}"
reemplazo="XX-XXXX-XXXX"

textoEncriptado=sub(patron,reemplazo,cadena)

print(f"Texto original {cadena}")
print(f"Texto enmascarado {textoEncriptado}")
```

Diccionarios

Diccionarios

- Son un tipo de estructura de datos que permite guardar un conjunto **no ordenado** de pares **clave:valor**, siendo las claves únicas dentro de un mismo diccionario
- Los diccionarios son mutables, es decir, es posible modificar su longitud, agregar o quitar elementos de el, de igual forma todos los valores almacenados en el diccionario pueden ser modificados

Diccionarios

- Al ser clave:valor **no se rigen** por la regla de los índices, porque todos los valores que se almacenen corresponderán a una clave o key
- **Todos los valores necesitan tener una key o clave, y cada clave necesita tener un valor**

```
nombreDiccionario={  
    clave1:valor1,  
    clave2:valor2,  
    claven: valorn  
}
```

Creación de diccionarios

- Se debe colocar el nombre del diccionario luego el igual, y todos los pares de **clave:valor** deben estar encerrados entre llaves {}
- Generalmente las claves o key **son strings**

```
persona={  
    "nombre": "Juan",  
    "edad": 24,  
    "nacionalidad": "Argentina",  
    "esProgramador": True,  
    "hobbies": ["Nadar", "Cantar"]  
}
```

Acceder a un valor

- Para acceder a un valor se debe utilizar corchetes, con la siguiente sintaxis:
 - **nombreDiccionario ["clave"]**

```
print(persona["nombre"])
```

Juan

Agregar nuevas claves

- Para agregar una nueva clave debemos también usar los paréntesis
 - **nombreDiccionario["nuevaClave"]="Valor"**

```
persona["clubFutbol"] = "Boca Juniors"
```

Modificar valor clave existente

Para modificar una clave existente es una sintaxis similar a crear una nueva clave, pero debemos colocar el nombre de la clave existente.

- **nombreDiccionario["clave"]="NuevoValor"**

```
persona={  
    "nombre": "Juan",  
    "edad": 24,  
    "nacionalidad": "Argentina",  
    "esProgramador": True,  
    "hobbies": ["Nadar", "Cantar"]  
}
```

```
persona["nombre"] = "Pedro"
```

```
print(persona)
```

```
{'nombre': 'Pedro', 'edad': 24, 'nacionalidad': 'Argentina',  
'esProgramador': True, 'hobbies': ['Nadar', 'Cantar']}
```

Acceso a clave inexistente

Si se desea acceder a una clave inexistente va a arrojar error **KeyError**

```
print(persona["nacimiento"])
```

Obtener cantidad de elementos

Al igual que en las listas se utiliza la función **len()**, enviando como parámetro el diccionario y devuelve la cantidad de pares **claves-valor** que contiene

```
largo=len(persona)  
  
print(f"El diccionario tiene {largo} pares clave-valor")
```

El diccionario tiene 5 pares clave-valor

Comparación de diccionarios

Se puede utilizar el operador de comparación **==** o de distinto **!=** para comparar si dos diccionarios son iguales

- **Dos diccionarios son iguales si contienen el mismo conjunto de pares clave:valor**, no solo en cantidad sino también en contenido.
- No importa el orden en el que estén
- Es la única comparación entre diccionarios permitida, cualquier otra arrojara error

Comparación de diccionarios - Ejemplo

```
definicion.py | Clase 01 | Clase 02 | Ejemplos | Ayuda  
diccionario1={  
    "clave2":"valor",  
    "clave1":1  
}  
  
diccionario2={  
    "clave1":1,  
    "clave2":"valor"  
}  
  
print(diccionario1==diccionario2)
```

Dos diccionarios iguales, pero con las claves en diferente orden

```
True
```

Comparación de diccionarios - Ejemplo

```
diccionario1={  
    "clave2":"valor",  
    "clave1":1  
}
```

Dos diccionarios con iguales claves pero diferentes valores

```
diccionario2=[{  
    "clave1":"1",  
    "clave2":"valor"  
}]
```

```
print(diccionario1==diccionario2)
```

False

Comparación de diccionarios - Ejemplo

```
diccionario1={  
    "clave2":"valor",  
    "clave1":1  
}
```

```
diccionario2={  
    "clave1":1,  
    "clave2":"valor"  
}
```

```
print(diccionario1<diccionario2)
```

Error al intentar comparar con operador diferente de igualdad o distinto, por ejemplo >

```
print(diccionario1<diccionario2)  
^^^^^^^^^^^^^^^^^^^^^^^^^
```

```
TypeError: '<' not supported between instances of 'dict'  
and 'dict'  
PC-GV-UADE\Julia\OneDrive\Desktop\OneDrive\UADE\Scen
```

Método get

- Este método devuelve el valor correspondiente a la clave que se envía.
- En caso que la clave no exista no lanza error, sino que devuelve None. Aunque existe la posibilidad de colocar un segundo argumento que seria el valor por defecto. Por ende si no existe, y pasamos segundo argumento devuelve el segundo.
- La sintaxis es
nombreDiccionario.get(“clave”, “valor_por_defecto”)
- valor_por_defecto es opcional

Método get

```
persona={  
    "nombre": "Juan",  
    "edad": 24,  
    "nacionalidad": "Argentina",  
    "esProgramador": True,  
    "hobbies": ["Nadar", "Cantar"]  
}
```

```
nombre= persona.get("nombre")
```

```
print(nombre)
```

Juan

Método get

```
persona={  
    "nombre": "Juan",  
    "edad": 24,  
    "nacionalidad": "Argentina",  
    "esProgramador": True,  
    "hobbies": ["Nadar", "Cantar"]  
}
```

```
apellido= persona.get("apellido", "Martin")
```

```
print(apellido)
```

Método get solicitando clave no existente, pero con valor por defecto

Martin

Método get con valor por defecto

```
persona={  
    "nombre": "Juan",  
    "edad": 24,  
    "nacionalidad": "Argentina",  
    "esProgramador": True,  
    "hobbies": ["Nadar", "Cantar"]  
}  
  
apellido= persona.get("apellido")  
  
print(apellido)
```

Método get solicitando clave no existente, sin valor por defecto

None

Recorrer un diccionario

Para recorrer tanto las **claves como los valores** se utiliza el método **items**

Unacademy / Cursos / Desarrollo / Clase 6 / Ejemplo.py

```
persona={  
    "nombre": "Juan",  
    "edad": 24,  
    "nacionalidad": "Argentina",  
    "esProgramador": True,  
    "hobbies": ["Nadar", "Cantar"]  
}
```

```
for clave, valor in persona.items():  
    print(clave, "-->", valor)
```

nombre --> Juan
edad --> 24
nacionalidad --> Argentina
esProgramador --> True
hobbies --> ['Nadar', 'Cantar']

Recorrer un diccionario

Si queremos obtener solo las claves utilizaremos el método **keys()**

```
persona={  
    "nombre": "Juan",  
    "edad": 24,  
    "nacionalidad": "Argentina",  
    "esProgramador": True,  
    "hobbies": ["Nadar", "Cantar"]  
}  
  
for clave in persona.keys():  
    print(clave)
```

```
nombre  
edad  
nacionalidad  
esProgramador  
hobbies
```

Recorrer un diccionario

Si queremos obtener solo los valores utilizaremos el método **values()**

```
persona={  
    "nombre": "Juan",  
    "edad": 24,  
    "nacionalidad": "Argentina",  
    "esProgramador": True,  
    "hobbies": ["Nadar", "Cantar"]  
}
```

```
for clave in persona.values():  
    print(clave)
```

```
Juan  
24  
Argentina  
True  
['Nadar', 'Cantar']
```

Verificar existencia

Se puede verificar la existencia de una **clave** en un diccionario con el operador **in**

```
aula01/1 - Curso Desgrana - Clase 0 - 3 - Ejemplo.py
personas = {
    "nombre": "Juan",
    "edad": 24,
    "nacionalidad": "Argentina",
    "esProgramador": True,
    "hobbies": ["Nadar", "Cantar"]
}

if "nacionalidad" in personas:
    print("Nacionalidad es una clave")
```

Nacionalidad es una clave

Limpiar diccionario

Se pueden eliminar todos los pares de clave -valor de un diccionario con el **metodo clear**.

nombreDiccionario.clear()

```
persona={  
    "nombre":"Juan",  
    "edad":24,  
    "nacionalidad":"Argentina",  
    "esProgramador":True,  
    "hobbies":["Nadar","Cantar"]}  
  
persona.clear()  
  
print(persona)
```

{}

Eliminar elemento

Para eliminar un elemento de un diccionario, se utiliza la función **del**. Si la clave a eliminar no existe, lanza un error

```
persona={  
    "nombre": "Juan",  
    "edad": 24,  
    "nacionalidad": "Argentina",  
    "esProgramador": True,  
    "hobbies": ["Nadar", "Cantar"]  
}
```

```
del persona["nacionalidad"]  
print(persona)
```

```
{'nombre': 'Juan', 'edad': 24, 'esProgramador': True, 'hobbies': ['Nadar', 'Cantar']}
```

Listas de diccionarios

Se pueden crear listas de diccionarios, y esto es muy útil cuando es necesario manejar una colección de elementos que requieren almacenar información estructurada y heterogénea.

Por ejemplo: una lista de ciudades donde cada ciudad tenga su nombre y su código postal

```
ciudades=[  
    {  
        "nombre": "La Plata",  
        "cp": 6237  
    },  
    {  
        "nombre": "america",  
        "cp": 1134  
    }  
]
```

Listas de diccionarios

Agregar un elemento a la lista de diccionario

```
ciudades.append({"nombre":"Rauch","cp":4444})
```

Acceso a un elemento

```
ciudades=[  
    {  
        "nombre":"La Plata",  
        "cp":6237  
    },  
    {  
        "nombre":"america",  
        "cp":1134  
    }  
]  
  
ciudades.append({"nombre":"Rauch","pepe":4444})  
print(ciudades[1]["cp"])
```

1134

Anidamiento

Es posible crear un diccionario cuyo valor contiene otro diccionario.

```
país={  
    "id":1,  
    "nombre":"Argentina",  
    "coordenadas":{  
        "latitud":65,  
        "longitud":-45  
    }  
}  
  
print(país["coordenadas"]["latitud"])
```

65

Ordenas las claves

Para ordenar un diccionario por sus claves se puede usar la función **sorted()** en combinación con el método **items()** del diccionario.

Luego con la función **dict()** lo convierte en diccionario

Ordenas las claves - Ejemplo

```
siglasSignificado={  
    "HTML": "Hipertext Markup Lenguaje",  
    "API": "Application Programming Interface",  
    "SQL": "Structured Query Language"  
}  
  
ordenado= dict(sorted(siglasSignificado.items()))  
  
print(ordenado)
```

```
{'API': 'Application Programming Interface', 'HTML': 'Hipertext Markup Lenguaje', 'SQL': 'Structured Query Language'}
```

Agregar un diccionario a otro diccionario

El método **update()** se utiliza para agregar elementos a un diccionario o para actualizar los valores existentes el diccionario con nuevos valores.

La sintaxis básica es:

diccionario.update({diccionario})

Recibe como parámetro otro diccionario que contiene pares **claves:valor** que se agregarán al diccionario existente

Ejemplo

```
diccionarioA={  
    "clave": "valor",  
    "clave2": "valor2"  
}
```

```
diccionarioB=[  
    "clave3": "valor3",  
    "clave4": "valor4"  
]
```

```
diccionarioA.update(diccionarioB)
```

```
print(diccionarioA)
```

```
{'clave': 'valor', 'clave2': 'valor2', 'clave3':  
'valor3', 'clave4': 'valor4'}
```

Práctica en clase

Realizar guías de:

- Funciones lambda
- Expresiones Regulares
- Diccionarios



Resumen de la clase

- Expresiones regulares en Python
- Diccionarios

Muchas gracias!

Consultas?

