Hoja de trucos del Módulo 1: Fundamentos de Python

Paquete/Método	Descripción	Ejemplo de Código
Comentarios	Los comentarios son líneas de texto que son ignoradas por el intérprete de Python al ejecutar el código	1. 1 1. # Este es un comentario Copied!
Concatenación	Combina (concatena) cadenas.	Sintaxis: 1. 1 1. cadena_concatenada = cadena1 + cadena2 Copied! Ejemplo: 1. 1 1. resultado = "Hola" + " John"</td> Copied!
Tipos de Datos	- Entero - Flotante - Booleano - Cadena	Ejemplo: 1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 8. 8 9. 9 10. 10 1. x=7 2. # Valor Entero 3. y=12.4 4. # Valor Flotante 5. es_valido = True 6. # Valor Booleano 7. es_valido = False 8. # Valor Booleano 9. Nombre = "John" 10. # Valor de Cadena Copied!
Indexación	Accede al carácter en un índice específico.	Ejemplo: 1. 1 2. 2 1. mis_cadenas="Hola" 2. caracter = mis_cadenas[0] Copied!
len()	Devuelve la longitud de una cadena.	Sintaxis: 1. 1 1. len(nombre_cadena) Copied!

Ejemplo:

```
1
2
```

```
mis_cadenas="Hola"
longitud = len(mis_cadenas)
```

Copied!

lower()	Convierte la cadena a minúsculas.	Ejemplo: `1` `2` `mis_cadenas="Hola"` `texto_minusc = mis_cadenas.lower()` Copied!
print()	Imprime el mensaje o variable dentro de `()`.	Ejemplo: `1` `2` `print("Hola, mundo")` `print(a+b)` Copied!
Operadores de Python	- Adición (+): Suma dos valores. - Sustracción (-): Resta un valor de otro. - Multiplicación (*): Multiplica dos valores. - División (/): Divide un valor por otro, devuelve un flotante. - División entera (//): Divide un valor por otro, devuelve el cociente como un entero. - Módulo (%): Devuelve el residuo después de la división.	Ejemplo: `1` `2` `3` `4` `5` `6` `7` `x = 9 y = 4` `resultado_suma= x + y # Adición` `resultado_resta= x - y # Sustracción` `resultado_multiplicacion= x * y # Multiplicación` `resultado_division= x / y # División` `resultado_division_entera= x // y # División entera` `resultado_modulo= x % y # Módulo</td>` Copied!
replace()	Reemplaza subcadenas.	Ejemplo: `1` `2` `mis_cadenas="Hola"` `nuevo_texto = mis_cadenas.replace("Hola", "Hola")` Copied!
Segmentación	Extrae una porción de la cadena.	Sintaxis: `1` `subcadena = nombre_cadena[inicio:fin]` Copied! Ejemplo:

		1. 1 1. mis_cadenas="Hola" subcadena = mis_cadenas[0:5] Copied! Ejemplo: 1. 1 2. 2 1. mis_cadenas="Hola" 2. texto_dividido = mis_cadenas.split(",") Copied!
split()	Divide la cadena en una lista basada en un delimitador.	
strip()	Elimina espacios en blanco al inicio y al final.	Ejemplo: 1. 1 2. 2 1. mis_cadenas="Hola" 2. recortado = mis_cadenas.strip() Copied!
upper()	Convierte la cadena a mayúsculas.	Ejemplo: 1. 1 2. 2 1. mis_cadenas="Hola" 2. texto_mayusc = mis_cadenas.upper() Copied!
Asignación de Variables	Asigna un valor a una variable.	Sintaxis: 1. 1 1. nombre_variable = valor Copied! Ejemplo: 1. 1 2. 2 1. nombre="John" # asignando John a la variable nombre 2. x = 5 # asignando 5 a la variable x Copied!

Hoja de trucos de estructuras de datos de Python

Lista

Paquete/Método	Descripción	Ejemplo de código
append()	El método `append()` se utiliza para agregar un elemento al final de una lista.	Sintaxis: 1. 1 1. list_name.append(element) Copied! Ejemplo: 1. 1 2. 2 1. fruits = ["manzana", "plátano", "naranja"] 2. fruits.append("mango") print(fruits) Copied!
copy()	El método `copy()` se utiliza para crear una copia superficial de una lista.	Ejemplo 1: 1. 1 2. 2 3. 3 1. my_list = [1, 2, 3, 4, 5] 2. new_list = my_list.copy() print(new_list) 3. # Salida: [1, 2, 3, 4, 5] Copied!
count()	El método `count()` se utiliza para contar el número de ocurrencias de un elemento específico en una lista en Python.	Ejemplo: 1. 1 2. 2 3. 3 1. my_list = [1, 2, 2, 3, 4, 2, 5, 2] 2. count = my_list.count(2) print(count) 3. # Salida: 4 Copied!
Crear una lista	Una lista es un tipo de dato incorporado que representa una colección ordenada y mutable de elementos. Las listas están encerradas en corchetes [] y los elementos están separados por comas.	Ejemplo: 1. 1 1. fruits = ["manzana", "plátano", "naranja", "mango"] Copied!
del	La declaración `del` se utiliza para eliminar un elemento de la lista. La declaración `del` elimina el elemento en el índice especificado.	Ejemplo: 1. 1 2. 2 3. 3 1. my_list = [10, 20, 30, 40, 50] 2. del my_list[2] # Elimina el elemento en el índice 2 print(my_list) 3. # Salida: [10, 20, 40, 50] Copied!
extend()	El método `extend()` se	Sintaxis:

	utiliza para agregar múltiples elementos a una lista. Toma un iterable (como otra lista, tupla o cadena) y agrega cada elemento del iterable a la lista original.	1 1. list_name.extend(iterable) Copied! Ejemplo: 1 2 3 4 1. fruits = ["manzana", "plátano", "naranja"] 2. more_fruits = ["mango", "uva"] 3. fruits.extend(more_fruits) 4. print(fruits) Copied!
Indexación	La indexación en una lista te permite acceder a elementos individuales por su posición. En Python, la indexación comienza en 0 para el primer elemento y llega hasta `length_of_list - 1`.	Ejemplo: 1 2 3 4 5 1. my_list = [10, 20, 30, 40, 50] 2. print(my_list[0]) 3. # Salida: 10 (accediendo al primer elemento) 4. print(my_list[-1]) 5. # Salida: 50 (accediendo al último elemento usando indexación negativa) Copied!
insert()	El método `insert()` se utiliza para insertar un elemento.	Sintaxis: 1 1. list_name.insert(index, element) Copied! Ejemplo: 1 2 3 1. my_list = [1, 2, 3, 4, 5] 2. my_list.insert(2, 6) 3. print(my_list) Copied!
Modificar una lista	Puedes usar la indexación para modificar o asignar nuevos valores a elementos específicos en la lista.	Ejemplo: 1 2 3 4 1. my_list = [10, 20, 30, 40, 50] 2. my_list[1] = 25 # Modificando el segundo elemento 3. print(my_list) 4. # Salida: [10, 25, 30, 40, 50] Copied!
pop()	El método `pop()` es otra forma de eliminar un elemento de una lista en Python. Elimina y devuelve el elemento en el índice especificado. Si	Ejemplo 1: 1 2 3 4 5 6 7 1. my_list = [10, 20, 30, 40, 50] 2. removed_element = my_list.pop(2) # Elimina y devuelve el elemento en el índice 2 3. print(removed_element) 4. # Salida: 30

	no proporcionas un índice al método `pop()`, eliminará y devolverá el último elemento de la lista por defecto.	5. 6. print(my_list) 7. # Salida: [10, 20, 40, 50] Copied! Ejemplo 2: 1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 1. my_list = [10, 20, 30, 40, 50] 2. removed_element = my_list.pop() # Elimina y devuelve el último elemento 3. print(removed_element) 4. # Salida: 50 5. 6. print(my_list) 7. # Salida: [10, 20, 30, 40] Copied!
remove()	Para eliminar un elemento de una lista. El método `remove()` elimina la primera ocurrencia del valor especificado.	Ejemplo: 1. 1 2. 2 3. 3 4. 4 1. my_list = [10, 20, 30, 40, 50] 2. my_list.remove(30) # Elimina el elemento 30 3. print(my_list) 4. # Salida: [10, 20, 40, 50] Copied!
reverse()	El método `reverse()` se utiliza para invertir el orden de los elementos en una lista.	Ejemplo 1: 1. 1 2. 2 3. 3 1. my_list = [1, 2, 3, 4, 5] 2. my_list.reverse() print(my_list) 3. # Salida: [5, 4, 3, 2, 1] Copied!
Segmentación	Puedes usar la segmentación para acceder a un rango de elementos de una lista.	Sintaxis: 1. 1 1. list_name[start:end:step] Copied! Ejemplo: 1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 8. 8 9. 9 10. 10 11. 11 12. 12 1. my_list = [1, 2, 3, 4, 5] 2. print(my_list[1:4]) 3. # Salida: [2, 3, 4] (elementos desde el índice 1 hasta 3) 4. 5. print(my_list[:3]) 6. # Salida: [1, 2, 3] (elementos desde el principio hasta el índice 2) 7. 8. print(my_list[2:]) 9. # Salida: [3, 4, 5] (elementos desde el índice 2 hasta el final) 10.

```
11. print(my_list[::2])
12. # Salida: [1, 3, 5] (cada segundo elemento)
```

Copied!

sort()	El método `sort()` se utiliza para ordenar los elementos de una lista en orden ascendente. Si deseas ordenar la lista en orden descendente, puedes pasar el argumento `reverse=True` al método `sort()`.	Ejemplo 1: 1. 1 2. 2 3. 3 4. 4 1. my_list = [5, 2, 8, 1, 9] 2. my_list.sort() 3. print(my_list) 4. # Salida: [1, 2, 5, 8, 9] Copied! Ejemplo 2: 1. 1 2. 2 3. 3 4. 4 1. my_list = [5, 2, 8, 1, 9] 2. my_list.sort(reverse=True) 3. print(my_list) 4. # Salida: [9, 8, 5, 2, 1] Copied!

Tupla

Paquete/Método	Descripción	Ejemplo de código
count()	El método count() para una tupla se utiliza para contar cuántas veces aparece un elemento especificado en la tupla.	Sintaxis: 1. 1 1. tuple.count(value) Copied! Ejemplo: 1. 1 2. 2 3. 3 1. fruits = ("manzana", "plátano", "manzana", "naranja") 2. print(fruits.count("manzana")) #Cuenta cuántas veces se encuentra manzana en la tupla. 3. #Salida: 2 Copied!
index()	El método index() en una tupla se utiliza para encontrar la primera ocurrencia de un valor especificado y devuelve su posición (índice). Si el valor no se encuentra, genera un ValueError.	Sintaxis: 1. 1 1. tuple.index(value) Copied! Ejemplo: 1. 1 2. 2 3. 3 1. fruits = ("manzana", "plátano", "naranja") 2. print(fruits[1]) #Devuelve el valor en el que está presente manzana. 3. #Salida: plátano Copied!
sum()	La función sum() en Python se puede usar para calcular la	Sintaxis: 1. 1 1. sum(tuple) Copied!

	suma de todos los elementos en una tupla, siempre que los elementos sean numéricos (enteros o flotantes).	Ejemplo: 1. 1 2. 2 3. 3 1. numbers = (10, 20, 5, 30) 2. print(sum(numbers)) 3. #Salida: 65 Copied!
min() y max()	Encuentra el elemento más pequeño (min()) o más grande (max()) en una tupla.	Ejemplo: 1. 1 2. 2 3. 3 4. 4 5. 5 1. numbers = (10, 20, 5, 30) 2. print(min(numbers)) 3. #Salida: 5 4. print(max(numbers)) 5. #Salida: 30 Copied!
len()	Obtén el número de elementos en la tupla usando len().	Sintaxis: 1. 1 1. len(tuple) Copied! Ejemplo: 1. 1 2. 2 3. 3 1. fruits = ("manzana", "plátano", "naranja") 2. print(len(fruits)) #Devuelve la longitud de la tupla. 3. #Salida: 3 Copied!

Hoja de trucos: Estructuras de datos de Python Parte-2

Diccionarios

Paquete/Método	Descripción	Ejemplo de código
Creating a Dictionary	Un diccionario es un tipo de dato incorporado que representa una colección de pares clave-valor. Los diccionarios están encerrados en llaves `{}`.	Ejemplo: 1 2 `dict_name = {} #Crea un diccionario vacío` `person = { "name": "John",  "age": 30, "city": "New York"}` Copied! Sintaxis: 1 `Value = dict_name["key_name"]` Copied!
Accediendo a valores	Puedes acceder a los valores en un diccionario usando sus correspondientes `keys`.	Ejemplo: 1 2 `name = person["name"]` `age = person["age"]` Copied! Sintaxis: 1 `dict_name[key] = value` Copied!
Agregar o modificar	Inserta un nuevo par clave-valor en el diccionario. Si la clave ya existe, el valor se actualizará; de lo contrario, se crea una nueva entrada.	Ejemplo: 1 2 `person["Country"] = "USA" # Se creará una nueva entrada.` `person["city"] = "Chicago"  # Actualiza el valor existente para la misma clave` Copied! Sintaxis: 1 `del dict_name[key]` Copied!
del	Elimina el par clave-valor especificado del diccionario. Lanza un `KeyError` si la clave no existe.	Ejemplo: 1 `del person["Country"]` Copied!
update()	El método `update()` fusiona el diccionario	Sintaxis: 1

	proporcionado en el diccionario existente, agregando o actualizando pares clave-valor.	`1. dict_name.update({key: value})` Copied! Ejemplo: `1. 1` `1. person.update({"Profession": "Doctor"})` Copied!
clear()	El método `clear()` vacía el diccionario, eliminando todos los pares clave-valor dentro de él. Después de esta operación, el diccionario sigue siendo accesible y se puede usar más adelante.	Sintaxis: `1. 1` `1. dict_name.clear()` Copied! Ejemplo: `1. 1` `1. grades.clear()` Copied!
existencia de clave	Puedes comprobar la existencia de una clave en un diccionario usando la palabra clave `in`.	Ejemplo: `1. 1` `2. 2` `1. if "name" in person:` `2. print("El nombre existe en el diccionario.")` Copied!
copy()	Crea una copia superficial del diccionario. El nuevo diccionario contiene los mismos pares clave-valor que el original, pero permanecen como objetos distintos en memoria.	Sintaxis: `1. 1` `1. new_dict = dict_name.copy()` Copied! Ejemplo: `1. 1` `2. 2` `1. new_person = person.copy()` `2. new_person = dict(person) # otra forma de crear una copia del diccionario` Copied!
keys()	Recupera todas las claves del diccionario y las convierte en una lista. Útil para iterar o procesar claves usando métodos de lista.	Sintaxis: `1. 1` `1. keys_list = list(dict_name.keys())` Copied! Ejemplo: `1. 1` `1. person_keys = list(person.keys())` Copied!
values()	Extrae todos los valores del diccionario y los convierte en una lista. Esta lista se puede usar para	Sintaxis: `1. 1` `1. values_list = list(dict_name.values())` Copied!

	procesamiento o análisis posterior.	Ejemplo: 1. 1 1. person_values = list(person.values()) Copied!
items()	Recupera todos los pares clave-valor como tuplas y las convierte en una lista de tuplas. Cada tupla consiste en una clave y su correspondiente valor.	Sintaxis: 1. 1 1. items_list = list(dict_name.items()) Copied! Ejemplo: 1. 1 1. info = list(person.items()) Copied!

Conjuntos

Paquete/Método	Descripción	Ejemplo de código
add()	Los elementos se pueden agregar a un conjunto usando el método `add()`. Los duplicados se eliminan automáticamente, ya que los conjuntos solo almacenan valores únicos.	Sintaxis: 1. 1 1. set_name.add(element) Copied! Ejemplo: 1. 1 1. fruits.add("mango") Copied!
clear()	El método `clear()` elimina todos los elementos del conjunto, resultando en un conjunto vacío. Actualiza el conjunto en su lugar.	Sintaxis: 1. 1 1. set_name.clear() Copied! Ejemplo: 1. 1 1. fruits.clear()</td> Copied!
copy()	El método `copy()` crea una copia superficial del conjunto. Cualquier modificación a la copia no afectará al conjunto original.	Sintaxis: 1. 1 1. new_set = set_name.copy() Copied! Ejemplo: 1. 1 1. new_fruits = fruits.copy() Copied!
Definiendo Conjuntos	Un conjunto es una colección desordenada de elementos únicos. Los conjuntos están encerrados en llaves `{}`. Son útiles para	Ejemplo: 1. 1 2. 2

	almacenar valores distintos y realizar operaciones de conjuntos.	1. empty_set = set() #Creando un conjunto vacío 2. Set fruits = {"apple", "banana", "orange"} Copied!
discard()	Usa el método `discard()` para eliminar un elemento específico del conjunto. Ignora si el elemento no se encuentra.	Sintaxis: 1. 1 1. set_name.discard(element) Copied! Ejemplo: 1. 1 1. fruits.discard("apple") Copied!
issubset()	El método `issubset()` verifica si el conjunto actual es un subconjunto de otro conjunto. Devuelve True si todos los elementos del conjunto actual están presentes en el otro conjunto, de lo contrario False.	Sintaxis: 1. 1 1. is_subset = set1.issubset(set2) Copied! Ejemplo: 1. 1 1. is_subset = fruits.issubset(colors) Copied!
issuperset()	El método `issuperset()` verifica si el conjunto actual es un superconjunto de otro conjunto. Devuelve True si todos los elementos del otro conjunto están presentes en el conjunto actual, de lo contrario False.	Sintaxis: is_superset = set1.issuperset(set2) Ejemplo: 1. 1 1. is_superset = colors.issuperset(fruits) Copied!
pop()	El método `pop()` elimina y devuelve un elemento arbitrario del conjunto. Lanza un `KeyError` si el conjunto está vacío. Usa este método para eliminar elementos cuando el orden no importa.	Sintaxis: 1. 1 1. removed_element = set_name.pop() Copied! Ejemplo: 1. 1 1. removed_fruit = fruits.pop() Copied!
remove()	Usa el método `remove()` para eliminar un elemento específico del conjunto. Lanza un `KeyError` si el elemento no se encuentra.	Sintaxis: 1. 1 1. set_name.remove(element) Copied! Ejemplo: 1. 1 1. fruits.remove("banana") Copied!

Operaciones de Conjuntos	Realiza varias operaciones en conjuntos: `unión`, `intersección`, `diferencia`, `diferencia simétrica`.	Sintaxis: 1. 1 2. 2 3. 3 4. 4 1. union_set = set1.union(set2) 2. intersection_set = set1.intersection(set2) 3. difference_set = set1.difference(set2) 4. sym_diff_set = set1.symmetric_difference(set2) Copied! Ejemplo: 1. 1 2. 2 3. 3 4. 4 1. combined = fruits.union(colors) 2. common = fruits.intersection(colors) 3. unique_to_fruits = fruits.difference(colors) 4. sym_diff = fruits.symmetric_difference(colors) Copied!
update()	El método `update()` agrega elementos de otro iterable al conjunto. Mantiene la unicidad de los elementos.	Sintaxis: 1. 1 1. set_name.update(iterable) Copied! Ejemplo: 1. 1 1. fruits.update(["kiwi", "grape"]) Copied!

Fundamentos de Programación en Python - Hoja de Referencia

Paquete/Método	Descripción	Ejemplo de Sintaxis y Código
AND	Devuelve `True` si tanto statement1 como statement2 son `True`. De lo contrario, devuelve `False`.	Sintaxis: `1` `statement1 and statement2` Copied! Ejemplo: `1 2 3 4 5 6 7 8 9` `marks = 90` `attendance_percentage = 87` `if marks >= 80 and attendance_percentage >= 85:` `    print("califica para honores")` `else:` `    print("No calificado para honores")` `# Salida = califica para honores` Copied!
Definición de Clase	Define un plano para crear objetos y definir sus atributos y comportamientos.	Sintaxis: `1` `class ClassName: # Atributos y métodos de la clase` Copied! Ejemplo: `1 2 3 4` `class Person:` `    def __init__(self, name, age):` `        self.name = name` `        self.age = age` Copied!
Definir Función	Una `función` es un bloque de código reutilizable que realiza una tarea específica o un conjunto de tareas cuando es llamado.	Sintaxis: `1` `def function_name(parameters): # Cuerpo de la función` Copied! Ejemplo: `1` `def greet(name): print("Hola,", name)` Copied!

Igual(==)	Verifica si dos valores son iguales.	Sintaxis: `1. 1` `1. variable1 == variable2` Copied! Ejemplo 1: `1. 1` `1. 5 == 5` Copied! devuelve True Ejemplo 2: `1. 1` `1. age = 25 age == 30` Copied! devuelve False
Bucle For	Un bucle `for` ejecuta repetidamente un bloque de código un número específico de iteraciones o sobre una secuencia de elementos (lista, rango, cadena, etc.).	Sintaxis: `1. 1` `1. for variable in sequence: # Código a repetir` Copied! Ejemplo 1: `1. 1` `2. 2` `1. for num in range(1, 10):` `2. print(num)` Copied! Ejemplo 2: `1. 1` `2. 2` `3. 3` `1. fruits = ["manzana", "plátano", "naranja", "uva", "kiwi"]` `2. for fruit in fruits:` `3. print(fruit)` Copied!
Llamada a Función	Una llamada a función es el acto de ejecutar el código dentro de la función utilizando los argumentos proporcionados.	Sintaxis: `1. 1` `1. function_name(arguments)` Copied! Ejemplo: `1. 1` `1. greet("Alice")` Copied!

Mayor o Igual que(>=)	Verifica si el valor de variable1 es mayor o igual a variable2.	Sintaxis: `1. 1` `1. variable1 >= variable2` Copied! Ejemplo 1: `1. 1` `1. 5 >= 5 y 9 >= 5` Copied! devuelve True Ejemplo 2: `1. 1` `2. 2` `3. 3` `1. quantity = 105` `2. minimum = 100` `3. quantity >= minimum` Copied! devuelve True
Mayor que(>)	Verifica si el valor de variable1 es mayor que variable2.	Sintaxis: `1. 1` `1. variable1 > variable2` Copied! Ejemplo 1: 9 > 6 devuelve True Ejemplo 2: `1. 1` `2. 2` `3. 3` `1. age = 20` `2. max_age = 25` `3. age > max_age` Copied! devuelve False
Sentencia If	Ejecuta el bloque de código `si` la condición es `True`.	Sintaxis: `1. 1` `1. if condition: #bloque de código para la sentencia if` Copied! Ejemplo: `1. 1` `2. 2` `1. if temperature > 30:` `2. print("¡Es un día caluroso!")`

		Copied! Sintaxis: ``` 1 2 3 4 5 6 7 8 ``` ``` if condition1: # Código si condition1 es True elif condition2: # Código si condition2 es True else: # Código si ninguna condición es True ```
If-Elif-Else	Ejecuta el primer bloque de código si condition1 es `True`, de lo contrario verifica condition2, y así sucesivamente. Si ninguna condición es `True`, se ejecuta el bloque else.	Copied! Ejemplo: ``` 1 2 3 4 5 6 7 8 9 ``` ``` score = 85 # Ejemplo de puntaje if score >= 90: print("¡Obtuviste una A!") elif score >= 80: print("¡Obtuviste una B!") else: print("Necesitas trabajar más duro.") # Salida = ¡Obtuviste una B! ```
		Copied! Sintaxis: ``` 1 2 ``` ``` if condition: # Código, si la condición es True else: # Código, si la condición es False ```
Sentencia If-Else	Ejecuta el primer bloque de código si la condición es `True`, de lo contrario el segundo bloque.	Copied! Ejemplo: ``` 1 2 3 4 ``` ``` if age >= 18: print("Eres un adulto.") else: print("Aún no eres un adulto.") ```
		Copied! Sintaxis: ``` 1 ``` ``` variable1 <= variable2 ```
Menor o Igual que(<=)	Verifica si el valor de variable1 es menor o igual a variable2.	

		Copied! Ejemplo 1: ``` 1. 1 ``` ``` 1. 5 <= 5 y 3 <= 5 ``` Copied! devuelve True Ejemplo 2: ``` 1. 1 2. 2 3. 3 ``` ``` 1. size = 38 2. max_size = 40 3. size <= max_size ``` Copied! devuelve True
Menor que(<)	Verifica si el valor de variable1 es menor que variable2.	Sintaxis: ``` 1. 1 ``` ``` 1. variable1 < variable2 ``` Copied! Ejemplo 1: ``` 1. 1 ``` ``` 1. 4 < 6 ``` Copied! devuelve True Ejemplo 2: ``` 1. 1 2. 2 3. 3 ``` ``` 1. score = 60 2. passing_score = 65 3. score < passing_score ``` Copied! devuelve True
Controles de Bucle	`break` sale del bucle prematuramente. `continue` salta el resto de la iteración actual y pasa a la siguiente iteración.	Sintaxis: ``` 1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 ``` ``` 1. for: # Código a repetir 2. if # declaración booleana 3. break 4. 5. for: # Código a repetir 6. if # declaración booleana 7. continue ```

Copied!

Ejemplo 1:

```
1
2
3
4
```

```
for num in range(1, 6):
    if num == 3:
        break
    print(num)
```

Copied!

Ejemplo 2:

```
1
2
3
4
```

```
for num in range(1, 6):
    if num == 3:
        continue
    print(num)
```

Copied!

Sintaxis:

```
1
```

```
!variable
```

Copied!

NOT	Devuelve `True` si la variable es `False`, y viceversa.	Ejemplo: `1` `!isLocked` Copied! devuelve True si la variable es False (es decir, desbloqueada).
No Igual(!=)	Verifica si dos valores no son iguales.	Sintaxis: `1` `variable1 != variable2` Copied! Ejemplo: `1` `2` `3` `a = 10` `b = 20` `a != b` Copied! devuelve True Ejemplo 2: `1` `2`

		1. count=0 2. count != 0 Copied! devuelve False
Creación de Objeto	Crea una instancia de una clase (objeto) utilizando el constructor de la clase.	Sintaxis: 1. 1 1. object_name = ClassName(arguments) Copied! Ejemplo: 1. 1 1. person1 = Person("Alice", 25) Copied!
OR	Devuelve `True` si statement1 o statement2 (o ambos) son `True`. De lo contrario, devuelve `False`.	Sintaxis: 1. 1 1. statement1 || statement2 Copied! Ejemplo: 1. 1 2. 2 1. "Invitación a la Fiesta de Despedida" 2. Grade = 12 grade == 11 or grade == 12 Copied! devuelve True
range()	Genera una secuencia de números dentro de un rango especificado.	Sintaxis: 1. 1 2. 2 3. 3 1. range(stop) 2. range(start, stop) 3. range(start, stop, step) Copied! Ejemplo: 1. 1 2. 2 3. 3 1. range(5) #genera una secuencia de enteros de 0 a 4. 2. range(2, 10) #genera una secuencia de enteros de 2 a 9. 3. range(1, 11, 2) #genera enteros impares de 1 a 9. Copied!
Sentencia Return	`Return` es una palabra clave utilizada para enviar un valor de vuelta desde una función a su llamador.	Sintaxis: 1. 1 1. return value Copied! Ejemplo:

```
1. 1
2. 2
```

```
1. def add(a, b): return a + b
2. result = add(3, 5)
```

Copied!

Sintaxis:

```
1. 1
2. 2
```

```
1. try: # Código que podría generar una excepción except
2. ExceptionType: # Código para manejar la excepción
```

Copied!

Bloque Try-Except	Intenta ejecutar el código en el bloque try. Si ocurre una excepción del tipo especificado, se ejecuta el código en el bloque except.	Ejemplo:

```
1. 1
2. 2
3. 3
4. 4
```

```
1. try:
2.     num = int(input("Introduce un número: "))
3. except ValueError:
4.     print("Entrada inválida. Por favor, introduce un número válido.")
```

Copied!

Sintaxis:

```
1. 1
2. 2
3. 3
```

```
1. try: # Código que podría generar una excepción except
2. ExceptionType: # Código para manejar la excepción
3. else: # Código a ejecutar si no ocurre ninguna excepción
```

Copied!

Try-Except con Bloque Else	El código en el bloque `else` se ejecuta si no ocurre ninguna excepción en el bloque try.	Ejemplo:

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
```

```
1. try:
2.     num = int(input("Introduce un número: "))
3. except ValueError:
4.     print("Entrada inválida. Por favor, introduce un número válido")
5. else:
6.     print("Introdujiste:", num)
```

Copied!

Try-Except con Bloque Finally	El código en el bloque `finally` siempre se ejecuta, independientemente de si ocurrió una excepción.	Sintaxis:

```
1. 1
2. 2
3. 3
```

```
1. try: # Código que podría generar una excepción except
2. ExceptionType: # Código para manejar la excepción
3. finally: # Código que siempre se ejecuta
```

Copied!

Ejemplo:

```
1. 1
```

		2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 1. try: 2. file = open("data.txt", "r") 3. data = file.read() 4. except FileNotFoundError: 5. print("Archivo no encontrado.") 6. finally: 7. file.close() Copied! Sintaxis: 1. 1 1. while condition: # Código a repetir Copied! Ejemplo: 1. 1 2. 2 1. count = 0 while count < 5: 2. print(count) count += 1 Copied!
Bucle While	Un bucle `while` ejecuta repetidamente un bloque de código mientras una condición especificada permanezca `True`.	

Trabajando con Datos en Python Cheat Sheet

Lectura y escritura de archivos

Paquete/Método	Descripción	
Modos de apertura de archivos	Diferentes modos para abrir archivos para operaciones específicas.	Sintaxis: r (lectura) w (escritura) a (agregar) + (actualización: lectura/escritura) b (binario, de lo contrario texto) 1. 1 1. Ejemplos: with open("data.txt", "r") as file: content = file.read() print(content) with open("output.txt", "w") as Copied!
Métodos de lectura de archivos	Diferentes métodos para leer el contenido de un archivo de varias maneras.	Sintaxis: 1. 1 2. 2 3. 3 1. file.readlines() # lee todas las líneas como una lista 2. readline() # lee la siguiente línea como una cadena 3. file.read() # lee todo el contenido del archivo como una cadena Copied! Ejemplo: 1. 1 2. 2 3. 3 4. 4 1. with open("data.txt", "r") as file: 2. lines = file.readlines() 3. next_line = file.readline() 4. content = file.read() Copied!
Métodos de escritura de archivos	Diferentes métodos de escritura para escribir contenido en un archivo.	Sintaxis: 1. 1 2. 2 1. file.write(content) # escribe una cadena en el archivo 2. file.writelines(lines) # escribe una lista de cadenas en el archivo Copied! Ejemplo: 1. 1 2. 2 3. 3 1. lines = ["Hola\n", "Mundo\n"] 2. with open("output.txt", "w") as file: 3. file.writelines(lines) Copied!
Iterando sobre líneas	Itera a través de cada línea en el archivo usando un `bucle`.	Sintaxis: 1. 1 1. for line in file: # Código para procesar cada línea Copied! Ejemplo: 1. 1 2. 2 1. with open("data.txt", "r") as file: 2. for line in file: print(line) Copied!
Open() y close()	Abrir un archivo, realizar operaciones y cerrar explícitamente el archivo usando el método close().	Sintaxis: 1. 1 2. 2 1. file = open(filename, mode) # Código que usa el archivo 2. file.close() Copied! Ejemplo: 1. 1 2. 2 3. 3 1. file = open("data.txt", "r") 2. content = file.read() 3. file.close() Copied!
with open()	Abrir un archivo	Sintaxis:

	usando un bloque with, asegurando el cierre automático del archivo después de su uso.	1. 1 1. with open(filename, mode) as file: # Código que usa el archivo Copied! Ejemplo: 1. 1 2. 2 1. with open("data.txt", "r") as file: 2. content = file.read() Copied!

Pandas

Paquete/Método	Descripción	Sintaxis y Ejemplo de Código
.read_csv()	Lee datos de un archivo `.CSV` y crea un DataFrame.	Sintaxis: dataframe_name = pd.read_csv("filename.csv") Ejemplo: df = pd.read_csv("data.csv")
.read_excel()	Lee datos de un archivo de Excel y crea un DataFrame.	Sintaxis: 1. 1 1. dataframe_name = pd.read_excel("filename.xlsx") Copied! Ejemplo: 1. 1 1. df = pd.read_excel("data.xlsx") Copied!
.to_csv()	Escribe el DataFrame en un archivo CSV.	Sintaxis: 1. 1 1. dataframe_name.to_csv("output.csv", index=False) Copied! Ejemplo: 1. 1 1. df.to_csv("output.csv", index=False) Copied!
Acceder a Columnas	Accede a una columna específica usando [] en el DataFrame.	Sintaxis: 1. 1 2. 2 1. dataframe_name["column_name"] # Accede a una sola columna 2. dataframe_name[["column1", "column2"]] # Accede a múltiples columnas Copied! Ejemplo: 1. 1 2. 2 1. df["edad"] 2. df[["nombre", "edad"]] Copied!
describe()	Genera un resumen estadístico de las columnas numéricas en el DataFrame.	Sintaxis: 1. 1 1. dataframe_name.describe() Copied! Ejemplo: 1. 1 1. df.describe() Copied!
drop()	Elimina filas o columnas específicas del DataFrame. axis=1 indica columnas. axis=0 indica filas.	Sintaxis: 1. 1 2. 2 1. dataframe_name.drop(["column1", "column2"], axis=1, inplace=True) 2. dataframe_name.drop(index=[row1, row2], axis=0, inplace=True) Copied! Ejemplo: 1. 1 2. 2

```
1. df.drop(["edad", "salario"], axis=1, inplace=True) # Eliminará columnas
2. df.drop(index=[5, 10], axis=0, inplace=True) # Eliminará filas
```

Copied!

Sintaxis:

```
1. 1
```

```
1. dataframe_name.dropna(axis=0, inplace=True)
```

Copied!

dropna() — Elimina filas con valores NaN faltantes del DataFrame. axis=0 indica filas.

Ejemplo:

```
1. 1
```

```
1. df.dropna(axis=0, inplace=True)
```

Copied!

Sintaxis:

```
1. 1
```

```
1. dataframe_name.duplicated()
```

Copied!

duplicated() — Valores o registros duplicados o repetitivos dentro de un conjunto de datos.

Ejemplo:

```
1. 1
```

```
1. duplicate_rows = df[df.duplicated()]
```

Copied!

Sintaxis:

```
1. 1
```

```
1. filtered_df = dataframe_name[(Condiciones)]
```

Copied!

Filtrar Filas — Crea un nuevo DataFrame con filas que cumplen condiciones específicas.

Ejemplo:

```
1. 1
```

```
1. filtered_df = df[(df["edad"] > 30) & (df["salario"] < 50000)
```

Copied!

Sintaxis:

```
1. 1
2. 2
```

```
1. grouped = dataframe_name.groupby(by, axis=0, level=None, as_index=True,
2. sort=True, group_keys=True, squeeze=False, observed=False, dropna=True)
```

Copied!

groupby() — Divide un DataFrame en grupos según criterios específicos, permitiendo la agregación, transformación o análisis posterior dentro de cada grupo.

Ejemplo:

```
1. 1
```

```
1. grouped = df.groupby(["categoría", "región"]).agg({"ventas": "suma"})
```

Copied!

Sintaxis:

```
1. 1
```

```
1. dataframe_name.head(n)
```

Copied!

head() — Muestra las primeras n filas del DataFrame.

Ejemplo:

```
1. 1
```

```
1. df.head(5)
```

Copied!

Sintaxis:

```
1. 1
```

```
1. import pandas as pd
```

Copied!

Importar pandas — Importa la biblioteca Pandas con el alias pd.

Ejemplo:

```
1. 1
```

```
1. import pandas as pd
```

Copied!

info()	Proporciona información sobre el DataFrame, incluidos los tipos de datos y el uso de memoria.	Sintaxis: `1. 1` `1. dataframe_name.info()` Copied! Ejemplo: `1. 1` `1. df.info()` Copied!
merge()	Combina dos DataFrames basándose en múltiples columnas comunes.	Sintaxis: `1. 1` `1. merged_df = pd.merge(df1, df2, on=["column1", "column2"])` Copied! Ejemplo: `1. 1` `1. merged_df = pd.merge(ventas, productos, on=["product_id", "category_id"])` Copied!
imprimir DataFrame	Muestra el contenido del DataFrame.	Sintaxis: `1. 1` `1. print(df) # o simplemente escribe df` Copied! Ejemplo: `1. 1` `2. 2` `1. print(df)` `2. df` Copied!
replace()	Reemplaza valores específicos en una columna por nuevos valores.	Sintaxis: `1. 1` `1. dataframe_name["column_name"].replace(old_value, new_value, inplace=True)` Copied! Ejemplo: `1. 1` `1. df["estado"].replace("En Progreso", "Activo", inplace=True)` Copied!
tail()	Muestra las últimas n filas del DataFrame.	Sintaxis: `1. 1` `1. dataframe_name.tail(n)` Copied! Ejemplo: `1. 1` `1. df.tail(5)` Copied!

Numpy

Paquete/Método	Descripción	Sintaxis y Ejemplo de Código
Importar NumPy	Importa la biblioteca NumPy.	Sintaxis: `1. 1` `1. import numpy as np` Copied! Ejemplo: `1. 1` `1. import numpy as np` Copied!
np.array()	Crea un array unidimensional o multidimensional,	Sintaxis: `1. 1`

```
2. 2
```

```
1. array_1d = np.array([valores de lista1]) # Array 1D
2. array_2d = np.array([[valores de lista1], [valores de lista2]]) # Array 2D
```

Copied!

Ejemplo:

```
1. 1
2. 2
```

```
1. array_1d = np.array([1, 2, 3]) # Array 1D
2. array_2d = np.array([[1, 2], [3, 4]]) # Array 2D
```

Copied!

Ejemplo:

Atributos de Array de Numpy	- Calcula la media de los elementos del array - Calcula la suma de los elementos del array - Encuentra el valor mínimo en el array - Encuentra el valor máximo en el array - Calcula el producto punto de dos arrays

```
1. 1
2. 2
3. 3
4. 4
5. 5
```

```
1. np.mean(array)
2. np.sum(array)
3. np.min(array)
4. np.max(array)
5. np.dot(array_1, array_2)
```

Copied!

Hoja de trucos: API y recopilación de datos

Paquete/Método	Descripción	Ejemplo de código
Accediendo al atributo del elemento	Accede al valor de un atributo específico de un elemento HTML.	Sintaxis: 1. 1 1. atributo = elemento[(atributo)] Copied! Ejemplo: 1. 1 1. href = link_element[(href)] Copied!
BeautifulSoup()	Analiza el contenido HTML de una página web utilizando BeautifulSoup. El tipo de analizador puede variar según el proyecto.	Sintaxis: 1. 1 1. soup = BeautifulSoup(html, (html.parser)) Copied! Ejemplo: 1. 1 1. html = (https://api.example.com/data) soup = BeautifulSoup(html, (html.parser)) Copied!
delete()	Envía una solicitud DELETE para eliminar datos o un recurso del servidor. Las solicitudes DELETE eliminan un recurso específico en el servidor.	Sintaxis: 1. 1 1. response = requests.delete(url) Copied! Ejemplo: 1. 1 1. response = requests.delete((https://api.example.com/delete)) Copied!
find()	Encuentra el primer elemento HTML que coincide con la etiqueta y atributos especificados.	Sintaxis: 1. 1 1. element = soup.find(tag, attrs) Copied! Ejemplo: 1. 1 1. first_link = soup.find((a), {(class): (link)}) Copied!
find_all()	Encuentra todos los elementos HTML que coinciden con la etiqueta y atributos especificados.	Sintaxis: 1. 1 1. elements = soup.find_all(tag, attrs) Copied! Ejemplo: 1. 1 1. all_links = soup.find_all((a), {(class): (link)})</td> Copied!
findChildren()	Encuentra todos los elementos hijos de un elemento HTML.	Sintaxis: 1. 1 1. children = element.findChildren() Copied! Ejemplo: 1. 1 1. child_elements = parent_div.findChildren() Copied!
get()	Realiza una solicitud GET	Sintaxis:

	para recuperar datos de una URL especificada. Las solicitudes GET se utilizan típicamente para leer datos de una API. La variable de respuesta contendrá la respuesta del servidor, que puedes procesar más adelante.	1. 1 1. response = requests.get(url) Copied! Ejemplo: 1. 1 1. response = requests.get((https://api.example.com/data)) Copied!
Headers	Incluye encabezados personalizados en la solicitud. Los encabezados pueden proporcionar información adicional al servidor, como tokens de autenticación o tipos de contenido.	Sintaxis: 1. 1 1. headers = {(HeaderName): (Value)} Copied! Ejemplo: 1. 1 1. base_url = (https://api.example.com/data) headers = {(Authorization): (Bearer YOUR_TOKEN)} response = requests.ge Copied!
Importar bibliotecas	Importa las bibliotecas de Python necesarias para el web scraping.	Sintaxis: 1. 1 1. from bs4 import BeautifulSoup Copied!
json()	Analiza datos JSON de la respuesta. Esto extrae y trabaja con los datos devueltos por la API. El método response.json() convierte la respuesta JSON en una estructura de datos de Python (generalmente un diccionario o lista).	Sintaxis: 1. 1 1. data = response.json() Copied! Ejemplo: 1. 1 2. 2 1. response = requests.get((https://api.example.com/data)) 2. data = response.json() Copied!
next_sibling()	Encuentra el siguiente elemento hermano en el DOM.	Sintaxis: 1. 1 1. sibling = element.find_next_sibling() Copied! Ejemplo: 1. 1 1. next_sibling = current_element.find_next_sibling() Copied!
parent	Accede al elemento padre en el Modelo de Objetos del Documento (DOM).	Sintaxis: 1. 1 1. parent = element.parent Copied! Ejemplo: 1. 1 1. parent_div = paragraph.parent Copied!
post()	Envía una solicitud POST	Sintaxis: 1. 1

	a una URL especificada con datos. Crea o actualiza solicitudes POST utilizando recursos en el servidor. El parámetro de datos contiene los datos que se enviarán al servidor, a menudo en formato JSON.	`1. response = requests.post(url, data)` Copied! Ejemplo: `1. 1` `1. response = requests.post((https://api.example.com/submit), data={(key): (value)})` Copied!
put()	Envía una solicitud PUT para actualizar datos en el servidor. Las solicitudes PUT se utilizan para actualizar un recurso existente en el servidor con los datos proporcionados en el parámetro de datos, típicamente en formato JSON.	Sintaxis: `1. 1` `1. response = requests.put(url, data)` Copied! Ejemplo: `1. 1` `1. response = requests.put((https://api.example.com/update), data={(key): (value)})` Copied!
Parámetros de consulta	Envía parámetros de consulta en la URL para filtrar o personalizar la solicitud. Los parámetros de consulta especifican condiciones o límites para los datos solicitados.	Sintaxis: `1. 1` `1. params = {(param_name): (value)}` Copied! Ejemplo: `1. 1` `2. 2` `3. 3` `1. base_url = "https://api.example.com/data"` `2. params = {"page": 1, "per_page": 10}` `3. response = requests.get(base_url, params=params)` Copied!
select()	Selecciona elementos HTML del HTML analizado utilizando un selector CSS.	Sintaxis: `1. 1` `1. element = soup.select(selector)` Copied! Ejemplo: `1. 1` `1. titles = soup.select((h1))` Copied!
status_code	Verifica el código de estado HTTP de la respuesta. El código de estado HTTP indica el resultado de la solicitud (éxito, error, redirección). El código de estado HTTP se puede usar para el manejo de errores y la toma de decisiones en tu código.	Sintaxis: `1. 1` `1. response.status_code` Copied! Ejemplo: `1. 1` `2. 2` `3. 3` `1. url = "https://api.example.com/data"` `2. response = requests.get(url)` `3. status_code = response.status_code` Copied!
etiquetas para find() y find_all()	Especifica cualquier etiqueta HTML válida como	Ejemplo de etiqueta: `1. 1` `2. 2` `3. 3`

	parámetro de etiqueta para buscar elementos de ese tipo. Aquí hay algunas etiquetas HTML comunes que puedes usar con el parámetro de etiqueta.	4. 4 5. 5 6. 6 7. 7 8. 8 9. 9 10. 10 1. - (a): Encontrar etiquetas de ancla (). 2. - (p): Encontrar etiquetas de párrafo ((p)). 3. - (h1), (h2), (h3), (h4), (h5), (h6): Encontrar etiquetas de encabezado del nivel 1 al 6 ((h1),n (h2)). 4. - (table): Encontrar etiquetas de tabla (). 5. - (tr): Encontrar etiquetas de fila de tabla (). 6. - (td): Encontrar etiquetas de celda de tabla ((td)). 7. - (th): Encontrar etiquetas de celda de encabezado de tabla ((td)). 8. - (img): Encontrar etiquetas de imagen ((img)). 9. - (form): Encontrar etiquetas de formulario ((form)). 10. - (button): Encontrar etiquetas de botón ((button)). Copied! Sintaxis: 1. 1 1. text = element.text Copied!
texto	Recupera el contenido de texto de un elemento HTML.	Ejemplo: 1. 1 1. title_text = title_element.text Copied!

Skills Network