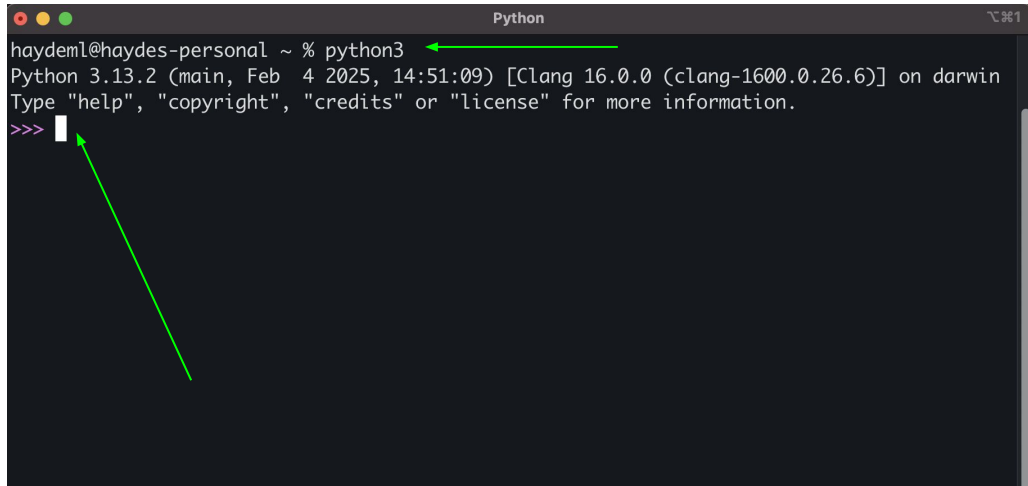




SINTAXIS - PYTHON

INTERACTUANDO CON PYTHON

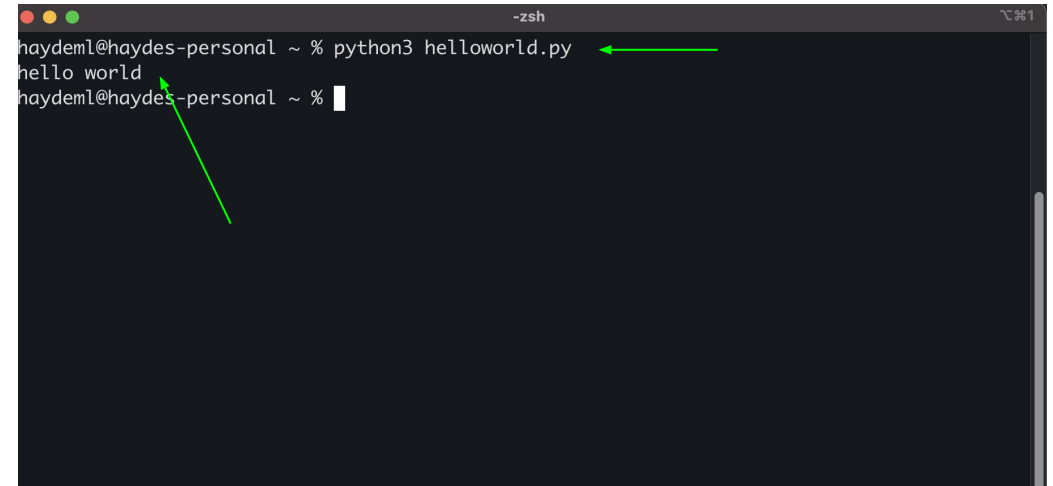
PARA ABRIR INTÉRPRETE:



A terminal window titled "Python" showing the command `python3` being executed. The output displays the Python version (3.13.2) and the environment (darwin). A green arrow points to the `python3` command, and another green arrow points to the prompt `>>>`.

```
haydeml@haydes-personal ~ % python3
Python 3.13.2 (main, Feb  4 2025, 14:51:09) [Clang 16.0.0 (clang-1600.0.26.6)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

PARA EJECUTAR UN SCRIPT GUARDADO:



A terminal window titled "-zsh" showing the command `python3 helloworld.py` being executed. The output displays "hello world". A green arrow points to the `python3 helloworld.py` command, and another green arrow points to the prompt `haydeml@haydes-personal ~ %`.

```
haydeml@haydes-personal ~ % python3 helloworld.py
hello world
haydeml@haydes-personal ~ %
```

EJEMPLO

MODULO SCRIPT

Ejemplo:

 mi_script.py

python

 Copy  Edit

```
print("Hola, este es mi primer script en Python.")
```

Para ejecutarlo en la terminal:

bash

 Copy  Edit

```
$ python3 mi_script.py
```

OPERACIONES CON CADENAS DE TEXTO

CONCATENACIÓN DE CADENAS

PUEDES UNIR CADENAS USANDO EL OPERADOR “+”:

```
nombre = "Python"
mensaje = "Bienvenido a " + nombre + "!"
print(mensaje) # "Bienvenido a Python!"
```

SI CONCATENAS NÚMEROS CON CADENAS, DEBES CONVERTIRLOS CON `str()`:

```
version = 3.12
mensaje = "Estás usando Python " + str(version)
print(mensaje) # "Estás usando Python 3.12"
```

OPERACIONES CON CADENAS DE TEXTO

MÉTODOS ÚTILES DE CADENAS

```
texto = "Hola Mundo"

print(texto.upper())    # 'HOLA MUNDO'
print(texto.lower())    # 'hola mundo'
print(texto.replace("Hola", "Adiós")) # 'Adiós Mundo'
print(texto.split())    # ['Hola', 'Mundo']
print(len(texto))       # 10
```

SLICING (REBANADO DE CADENAS)

```
texto = "Python"

print(texto[0])         # 'P'   (primer carácter)
print(texto[-1])        # 'n'   (último carácter)
print(texto[0:3])        # 'Pyt' (desde el índice 0 hasta el 2)
```

OPERACIONES CON LISTAS

LAS LISTAS SON SECUENCIAS DE ELEMENTOS ORDENADOS DE CUALQUIER TIPO.

PARA ESCRIBIRLAS SE UTILIZA UNA SECUENCIA DE EXPRESIONES SEPARADAS POR COMAS ENTRE CORCHETES:

```
mi_lista = [1, 2, 3, "Python", True]
print(mi_lista[0])    # 1
print(mi_lista[-1])   # True
```

MÉTODOS ÚTILES DE LAS LISTAS:

```
mi_lista.append(4)      # Agrega 4 al final
mi_lista.insert(2, 99)  # Inserta 99 en la posición 2
mi_lista.remove(3)      # Elimina el primer 3 encontrado
mi_lista.pop()          # Elimina el último elemento
mi_lista.sort()         # Ordena los elementos (si son comparables)
mi_lista.reverse()      # Invierte el orden
print(mi_lista.count(1)) # Cuenta las veces que aparece 1
print(len(mi_lista))    # Longitud de la lista
```

OPERACIONES CON LISTAS

LAS LISTAS PUEDEN CONTENER ENTEROS, BOOLEANOS, CADENAS DE TEXTO, FLOTANTES O INCLUSIVE LISTAS.

CADA ELEMENTO DE LA LISTA CUENTA CON UN ÍNDICE INICIANDO EN 0 PARA EL PRIMER ELEMENTO.

```
lista = [1, "dos", False, [45, "cien"]]
```

AHORA, ¿CÓMO HARÍA PARA ACCEDER A UN ELEMENTO DE UNA LISTA QUE ESTÁ DENTRO DE OTRA LISTA?

```
num = lista[3][0]
```

TAMBIÉN PODEMOS TRABAJAR CON ÍNDICES NEGATIVOS, MUY ÚTILES A LA HORA DE SABER NUESTRO ÚLTIMO ELEMENTO DE LA LISTA Ó RECORRER UNA LISTA DE ATRÁS HACIA ADELANTE.

```
nombres = ["Diego", "Carlos", "Martin", "Lorena", "Natalia"]  
print nombres[-1]
```

EL MÉTODO SPLIT

```
>>>"Hola mi nombre es Diego".split()  
["Hola", "mi", "nombre", "es", "Diego"]
```

OPERACIONES CON TUPLAS

LAS TUPLAS SON SECUENCIAS DE ELEMENTOS ORDENADOS DE CUALQUIER TIPO QUE NO SE PUEDEN MODIFICAR.

PARA ESCRIBIRLAS SE UTILIZA UNA SECUENCIA DE EXPRESIONES SEPARADAS POR COMAS ENTRE PARÉNTESIS:

```
mi_tupla = (1, 2, 3, "Python")  
print(mi_tupla[1]) # 2
```

MÉTODOS ÚTILES DE LAS TUPLAS:

```
print(mi_tupla.count(2)) # Cuenta las veces que aparece 2  
print(mi_tupla.index("Python")) # Índice de "Python"
```


OPERACIONES CON TUPLAS

¿ENTONCES PARA QUÉ SIRVEN LAS TUPLAS?

- LAS TUPLAS SON MÁS RÁPIDAS QUE LAS LISTAS.
- SI ESTAMOS DEFINIENDO UN CONJUNTO CONSTANTE DE VALORES Y TODO LO QUE VAMOS A HACER CON ÉL ES RECORRERLA, UTILIZAMOS UNA TUPLA EN LUGAR DE UNA LISTA.
- LAS CLAVES DE UN DICCIONARIO PUEDEN SER ENTEROS, CADENAS Y “ALGUNOS OTROS TIPOS”; LAS TUPLAS SON UNO DE ESTOS TIPOS, PUEDEN UTILIZARSE COMO CLAVES EN UN DICCIONARIO, PERO LAS LISTAS NO.
- LAS TUPLAS PUEDEN CONVERTIRSE EN LISTAS, Y VICEVERSA.
- LA FUNCIÓN INCORPORADA TUPLE TOMA UNA LISTA Y DEVUELVE UNA TUPLA CON LOS MISMOS ELEMENTOS, Y LA FUNCIÓN LIST TOMA UNA TUPLA Y DEVUELVE UNA LISTA.

OPERACIONES CON DICCIONARIOS

LOS DICCIONARIOS SON ESTRUCTURAS DE DATOS CLAVE-VALOR, IDEALES PARA ALMACENAR INFORMACIÓN SEMI-ESTRUCTURADA

PARA ESCRIBIRLAS SE UTILIZA UNA SECUENCIA DE EXPRESIONES “CLAVE”:“VALOR” SEPARADAS POR COMAS Y ENTRE LLAVES:

```
mi_dict = {"nombre": "Python", "versión": 3.12, "activo": True}
print(mi_dict["nombre"]) # "Python"
```

MÉTODOS ÚTILES DE LOS DICCIONARIOS:

```
mi_dict["año"] = 2024 # Agregar nueva clave
print(mi_dict.keys()) # Devuelve las claves
print(mi_dict.values()) # Devuelve los valores
print(mi_dict.items()) # Devuelve pares clave-valor
mi_dict.pop("activo") # Elimina una clave específica
mi_dict.clear() # Vacía el diccionario
```

OPERACIONES CON DICCIONARIOS

PERMITEN ACCESO RÁPIDO MEDIANTE CLAVES EN LUGAR DE ÍNDICES, PUEDEN MODIFICARSE DESPUÉS DE SU CREACIÓN Y NO ACEPTAN CLAVES DUPLICADAS.

SI INTENTAMOS ACCEDER A UNA CLAVE INEXISTENTE, PYTHON LANZA UN ERROR:

```
materias = {}  
materias["lunes"] = [6103, 7540]  
materias["martes"] = [6201]  
materias["miércoles"] = [6103, 7540]  
materias["jueves"] = []  
materias["viernes"] = [6201]  
  
print(materias["domingo"]) # KeyError
```

PARA EVITARLO, USAMOS GET() QUE DEVUELVE UN VALOR POR DEFECTO SI LA CLAVE NO EXISTE:

```
print(materias.get("domingo", "no existe")) # "no existe"
```

HAY DOS MANERAS PARA RECORRER (ITERAR) UN DICCIONARIO:

POR CLAVE:

```
for dia in materias:  
    print(dia, ":", materias[dia])
```

POR CLAVE-VALOR CON ITEMS():

```
for dia, codigos in materias.items():  
    print(dia, ":", codigos)
```

OPERACIONES CON DICCIONARIOS

PARA VERIFICAR SI UNA CLAVE EXISTE:

```
d = {'x': 12, 'y': 7}
if "x" in d:
    print(d["x"]) # 12
```

EFICIENCIA EN LA BÚSQUEDA:

- PYTHON USA UN ALGORITMO **HASH** PARA BUSCAR ELEMENTOS EN UN DICCIONARIO
- A DIFERENCIA DE LAS LISTAS, EL TIEMPO DE BÚSQUEDA ES INDEPENDIENTE DEL TAMAÑO
- POR ESTA RAZÓN, LAS CLAVES DEBEN SER INMUTABLES

UN ALGORITMO HASH ES COMO UN BUZÓN DE CORREOS:
CADA CARTA (VALOR) SE GUARDA EN UN BUZÓN ESPECÍFICO
USANDO EL NOMBRE DEL DESTINATARIO (CLAVE). EN LUGAR
DE BUSCAR EN TODOS LOS BUZONES, EL CARTERO USA UNA
FÓRMULA MÁGICA (HASH) QUE LE DICE EXACTAMENTE
DÓNDE COLOCAR O ENCONTRAR CADA CARTA, HACIÉNDOLO
RÁPIDO Y EFICIENTE.