Fundamentos de Python

Tu primera inmersión en programación

Python Full Stack - Clase 14

🎯 Objetivos de Hoy

¿Qué aprenderemos?

- 📊 Comprender variables y tipos de datos en Python
- 🔢 Trabajar con números, cadenas y booleanos
- 📦 Dominar estructuras de datos: listas, tuplas y diccionarios
- 🔀 Controlar el flujo con condicionales
- 🔁 Repetir acciones con bucles for y while
- 💪 Resolver problemas prácticos con Python

💾 Variables y Tipos de Datos

💾 ¿Qué es una Variable?

¿Cuándo necesito esto?

Escenario: "Necesito guardar información para usarla después"

Analogía de la vida real:

- Una variable es como una **caja con etiqueta** 📦
- La etiqueta es el **nombre** de la variable
- Dentro de la caja está el **valor** que guardamos
- Podemos cambiar lo que hay dentro de la caja cuando queramos

💾 Crear Variables en Python

Sintaxis simple:

```
# Crear una variable es muy fácil
nombre = "Ana"
edad = 25
altura = 1.65
estudiante = True
```

Principios esenciales:

- No necesitas declarar el tipo (Python lo detecta automáticamente)
- Usa `=` para asignar un valor
- Los nombres de variables usan minúsculas y guiones bajos: `mi_variable`
- No uses palabras reservadas como: `if`, `for`, `class`

📊 Tipos de Datos Primitivos

Los bloques básicos de construcción:

Tipo	Descripción	Ejemplo
Boolean	Verdadero o Falso	`True`, `False`
Integer	Números enteros	`42`, `-10`, `0`
Float	Números decimales	`3.14`, `1.5`, `-0.5`
String	Texto/cadenas	`"Hola"`, `'Python'`

💻 Ejemplo: Tipos de Datos Primitivos

```
# Boolean - para decisiones
es_mayor_edad = True
tiene_licencia = False

# Integer - números enteros
edad = 25
cantidad_estudiantes = 30
temperatura = -5

# Float - números con decimales
altura = 1.75
precio = 19.99
pi = 3.1416

# String - texto
nombre = "María"
mensaje = 'Hola mundo'
direccion = "Av. Principal 123"
```

🔍 Función type() - Descubrir Tipos

¿Cuándo necesito esto?

Escenario: "No estoy seguro qué tipo de dato es esta variable"

```
edad = 25
print(type(edad))  # Imprime: <class 'int'>

altura = 1.75
print(type(altura))  # Imprime: <class 'float'>

nombre = "Ana"
print(type(nombre))  # Imprime: <class 'str'>

activo = True
print(type(activo))  # Imprime: <class 'bool'>
```

💡 **Tip:** Usa `type()` cuando tengas dudas sobre un tipo de dato

🔢 Números en Python

🔢 Tipos de Números

1. **int** - Números enteros (sin decimales)

```
edad = 25
año = 2025
temperatura = -10
```

2. **float** - Números con decimales

```
altura = 1.75
precio = 99.99
pi = 3.1416
```

🔢 Operaciones con Números

Operadores matemáticos básicos:

```
# Suma
resultado = 10 + 5  # 15

# Resta
resultado = 10 - 3  # 7

# Multiplicación
resultado = 4 * 5  # 20

# División (siempre devuelve float)
resultado = 10 / 3  # 3.333...

# División entera (sin decimales)
resultado = 10 // 3  # 3

# Módulo (resto de la división)
resultado = 10 % 3  # 1

# Potencia
resultado = 2 ** 3  # 8 (2 elevado a la 3)
```

🔄 Conversión de Tipos

¿Cuándo necesito esto?

Escenario: "Necesito convertir un número entero a decimal o viceversa"

```
# De entero a decimal
entero = 123
decimal = float(entero)
print(decimal)  # 123.0

# De decimal a entero (pierde los decimales)
decimal = 22.8
entero = int(decimal)
print(entero)  # 22

# De texto a número
texto_numero = "50"
numero = int(texto_numero)
print(numero + 10)  # 60
```

⚠️ **Importante:** No puedes convertir texto no numérico a número

🎲 Números Aleatorios

Generar números al azar:

```python
import random  # Primero importamos la librería

# Número aleatorio entre 1 y 10 (incluye ambos)
numero = random.randint(1, 10)
print(numero)  # Ejemplo: 7

# Número aleatorio entre 5 y 100
numero = random.randint(5, 100)
print(numero)  # Ejemplo: 47

# Número decimal aleatorio entre 0 y 1
decimal = random.random()
print(decimal)  # Ejemplo: 0.7234...
```

💡 **Uso común:** Juegos, simulaciones, sorteos

Strings/Cadenas

📝 ¿Qué es un String?

¿Cuándo necesito esto?

Escenario: "Necesito trabajar con texto: nombres, mensajes, descripciones"

Definición: Una cadena es una secuencia de caracteres (letras, números, símbolos)

```
# Comillas simples o dobles (funciona igual)
nombre = "María"
apellido = 'González'

# Texto largo
mensaje = "Bienvenidos al curso de Python"

# Puede incluir números y símbolos
codigo = "ABC-123"
email = "maria@email.com"
```

✨ Concatenar Cadenas

Método 1: Usando comas con print()

```
nombre = "Carlos"
edad = 28
# Con coma (print agrega espacio automático)
print("Hola", nombre)  # Hola Carlos
print("Tengo", edad, "años")  # Tengo 28 años
```

Método 2: Usando el operador +

```
nombre = "Carlos"
# Con + (sin espacios automáticos)
print("Hola " + nombre)  # Hola Carlos
# ⚠️ No funciona con números directamente
edad = 28
# print("Tengo " + edad)  # ERROR!
```

🔄 Conversión de Tipos con Strings

¿Cuándo necesito esto?

Escenario: "Quiero unir texto con números usando +"

```
# Problema: no puedes sumar string + número
edad = 25
# mensaje = "Tengo " + edad + " años"  # ❌ ERROR!

# Solución 1: Convertir número a string
edad = 25
mensaje = "Tengo " + str(edad) + " años"
print(mensaje)  # Tengo 25 años ✅

# Solución 2: Usar comas (más fácil)
print("Tengo", edad, "años")  # Tengo 25 años ✅
```

🎨 Interpolación de Cadenas - f-strings

La forma moderna y recomendada (Python 3.6+):

```
nombre = "Ana"
edad = 30
ciudad = "Santiago"

# f-string: coloca f antes de las comillas
mensaje = f"Me llamo {nombre}, tengo {edad} años y vivo en {ciudad}"
print(mensaje)
# Imprime: Me llamo Ana, tengo 30 años y vivo en Santiago

# Puedes hacer operaciones dentro de {}
precio = 100
descuento = 20
print(f"Precio final: ${precio - descuento}")
# Imprime: Precio final: $80
```

💡 **Recomendación:** Usa f-strings, es lo más limpio y legible

📝 Interpolación - Método .format()

Forma anterior (aún válida):

```
nombre = "Luis"
edad = 28

# .format() con llaves vacías {}
mensaje = "Me llamo {} y tengo {} años".format(nombre, edad)
print(mensaje)
# Imprime: Me llamo Luis y tengo 28 años

# Puedes cambiar el orden con índices
mensaje = "Tengo {1} años y me llamo {0}".format(nombre, edad)
print(mensaje)
# Imprime: Tengo 28 años y me llamo Luis
```

🛠️ Métodos Útiles de Strings

```python
texto = "Hola Mundo"

# Convertir a mayúsculas
print(texto.upper())  # HOLA MUNDO

# Convertir a minúsculas
print(texto.lower())  # hola mundo

# Contar ocurrencias
print(texto.count("o"))  # 2

# Dividir en palabras (crea una lista)
palabras = texto.split()
print(palabras)  # ['Hola', 'Mundo']

# Reemplazar texto
nuevo = texto.replace("Mundo", "Python")
print(nuevo)  # Hola Python

# Verificar si empieza con algo
print(texto.startswith("Hola"))  # True
```

💪 Ejercicio: ¡Hola Mundo!

¡Hora de practicar!

Crea un archivo `hola_mundo.py` con:

1. Imprime "Hola, mundo"
2. Crea una variable `nombre` con tu nombre e imprime "Hola, [tu nombre]"
3. Crea una variable `edad` con tu edad e imprime "Tengo [edad] años"
4. Crea dos variables `comida1` y `comida2` e imprime "Me encanta comer [comida1] y [comida2]" usando f-string

Pistas:

- Usa `print()` para mostrar en pantalla
- Recuerda usar f-strings para interpolar: `f"Texto {variable}"`
- Practica diferentes formas de concatenar

⏱️ **Tiempo:** 10 minutos

📦 Estructuras de Datos

Listas, Tuplas y Diccionarios

📦 ¿Qué son las Estructuras de Datos?

¿Cuándo necesito esto?

Escenario: "Necesito guardar múltiples valores relacionados"

Analogía de la vida real:

- **Lista** 📝 = Lista del supermercado (puedes agregar/quitar items)
- **Tupla** 🔒 = Coordenadas GPS (no cambian)
- **Diccionario** 📖 = Agenda telefónica (nombre → teléfono)

Principio esencial:
Las estructuras de datos nos permiten organizar y acceder a múltiples valores de manera eficiente.

📋 Listas

📋 ¿Qué es una Lista?

Características principales:

- ✅ **Mutable** - Puedes modificarla después de crearla
- ✅ **Ordenada** - Los elementos tienen posición (índice)
- ✅ **Dinámica** - Crece y decrece según necesites
- ✅ **Versátil** - Puede contener diferentes tipos de datos

```
# Crear listas
lista_vacia = []
numeros = [1, 2, 3, 4, 5]
frutas = ["manzana", "banana", "naranja"]
mixta = [1, "texto", True, 3.14]  # ¡Diferentes tipos!
```

📋 Acceder a Elementos de una Lista

Índices comienzan en 0:

```
frutas = ["manzana", "banana", "naranja", "uva"]

# Índice positivo (desde el inicio)
print(frutas[0])  # manzana (primer elemento)
print(frutas[1])  # banana
print(frutas[2])  # naranja

# Índice negativo (desde el final)
print(frutas[-1])  # uva (último elemento)
print(frutas[-2])  # naranja (penúltimo)

# Modificar un elemento
frutas[1] = "pera"
print(frutas)  # ['manzana', 'pera', 'naranja', 'uva']
```

📊 **Visualización:** `[0: manzana][1: banana][2: naranja][3: uva]`

📋 Métodos Importantes de Listas

```python
frutas = ["manzana", "banana"]

# .append() - Agregar al final
frutas.append("naranja")
print(frutas)  # ['manzana', 'banana', 'naranja']

# .insert() - Insertar en posición específica
frutas.insert(1, "pera")  # Índice 1
print(frutas)  # ['manzana', 'pera', 'banana', 'naranja']

# .pop() - Eliminar y retornar el último
ultima = frutas.pop()
print(ultima)  # naranja
print(frutas)  # ['manzana', 'pera', 'banana']

# .pop(índice) - Eliminar elemento específico
frutas.pop(1)
print(frutas)  # ['manzana', 'banana']
```

📋 Más Métodos de Listas

```
numeros = [3, 1, 4, 1, 5, 9, 2]

# .sort() - Ordenar la lista (modifica original)
numeros.sort()
print(numeros)  # [1, 1, 2, 3, 4, 5, 9]

# .reverse() - Invertir orden
numeros.reverse()
print(numeros)  # [9, 5, 4, 3, 2, 1, 1]

# .count() - Contar ocurrencias
print(numeros.count(1))  # 2

# .index() - Encontrar posición
print(numeros.index(5))  # Índice donde está el 5

# len() - Longitud de la lista
print(len(numeros))  # 7
```

📋 Slicing - Rebanadas de Listas

Obtener partes de una lista:

```
numeros = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

# [inicio:fin] - fin no se incluye
print(numeros[2:5])  # [2, 3, 4]
# [:fin] - Desde el inicio hasta fin
print(numeros[:4])  # [0, 1, 2, 3]
# [inicio:] - Desde inicio hasta el final
print(numeros[6:])  # [6, 7, 8, 9]
# [inicio:fin:paso] - Con saltos
print(numeros[0:10:2])  # [0, 2, 4, 6, 8] (de 2 en 2)
# [::-1] - Invertir lista
print(numeros[::-1])  # [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

📋 Combinar y Multiplicar Listas

```
# Sumar listas (concatenar)
lista1 = [1, 2, 3]
lista2 = [4, 5, 6]
lista_completa = lista1 + lista2
print(lista_completa)  # [1, 2, 3, 4, 5, 6]

# Multiplicar lista (repetir)
numeros = [1, 2, 3]
repetida = numeros * 3
print(repetida)  # [1, 2, 3, 1, 2, 3, 1, 2, 3]

# Ejemplo práctico
separador = ["---"] * 5
print(separador)  # ['---', '---', '---', '---', '---']
```

🔒 Tuplas

🔒 ¿Qué es una Tupla?

Características principales:

- 🔒 **Inmutable** - NO puedes modificarla después de crearla
- ✅ **Ordenada** - Los elementos tienen posición
- ⚡ **Rápida** - Más eficiente que las listas
- 📦 **Agrupación** - Ideal para datos que van juntos

```
# Crear tuplas (con paréntesis o sin ellos)
coordenadas = (10, 20)
persona = ("Ana", 25, "Santiago")
colores = "rojo", "verde", "azul"  # También es tupla

# Tupla vacía
vacia = ()
```

🔒 Acceder a Elementos de Tuplas

```python
persona = ("Carlos", 30, "Desarrollador", "Chile")

# Acceder por índice (igual que listas)
print(persona[0])  # Carlos
print(persona[1])  # 30
print(persona[-1])  # Chile (último)

# Intentar modificar causa error
# persona[0] = "Luis"  # ❌ ERROR: 'tuple' object does not support item assignment

# Puedes "desempaquetar" tuplas
nombre, edad, profesion, pais = persona
print(f"{nombre} tiene {edad} años")  # Carlos tiene 30 años
```

🔒 ¿Cuándo Usar Tuplas?

Casos de uso comunes:

1. **Datos que no deben cambiar**

```
DIAS_SEMANA = ("Lun", "Mar", "Mié", "Jue", "Vie", "Sáb", "Dom")
```

2. **Retornar múltiples valores de una función**

```
def obtener_coordenadas():
    return (10.5, -33.4)  # latitud, longitud
```

3. **Claves de diccionarios** (las listas no pueden ser claves)

```
ubicaciones = {
    (10, 20): "Punto A",
    (30, 40): "Punto B"
}
```

🔒 Métodos y Funciones para Tuplas

```
numeros = (1, 2, 3, 2, 4, 2, 5)

# .count() - Contar ocurrencias
print(numeros.count(2))  # 3
# .index() - Encontrar posición
print(numeros.index(4))  # 4
# len() - Longitud
print(len(numeros))  # 7
# min(), max(), sum() - Con tuplas numéricas
print(min(numeros))  # 1
print(max(numeros))  # 5
print(sum(numeros))  # 19
# Convertir entre lista y tupla
lista = list(numeros)  # Tupla → Lista
tupla = tuple(lista)   # Lista → Tupla
```

📖 Diccionarios

📖 ¿Qué es un Diccionario?

¿Cuándo necesito esto?

Escenario: "Necesito asociar información relacionada: nombre con edad, país con capital"

Analogía de la vida real:
Un diccionario es como una **agenda telefónica** 📱

- **Clave (key)**: El nombre de la persona
- **Valor (value)**: El número de teléfono

```
estudiante = {
    "nombre": "María",
    "edad": 22,
    "curso": "Python",
    "activo": True
}
```

📖 Crear y Acceder a Diccionarios

```
# Diccionario vacío
persona = {}

# Agregar elementos
persona["nombre"] = "Juan"
persona["edad"] = 28
persona["ciudad"] = "Lima"

print(persona)
# {'nombre': 'Juan', 'edad': 28, 'ciudad': 'Lima'}

# Acceder a valores por clave
print(persona["nombre"])  # Juan
print(persona["edad"])    # 28

# Modificar valor existente
persona["edad"] = 29
print(persona["edad"])  # 29
```

📖 Diccionario con Datos Complejos

```
# Diccionario puede contener listas, tuplas u otros diccionarios
curso = {
    "nombre": "Python Full Stack",
    "duracion": 16,  # semanas
    "modalidad": "online",
    "tecnologias": ["Python", "Flask", "MySQL"],
    "instructor": {
        "nombre": "Carlos",
        "experiencia": 5
    }
}

# Acceder a datos anidados
print(curso["tecnologias"][0])  # Python
print(curso["instructor"]["nombre"])  # Carlos
```

📖 Métodos Importantes de Diccionarios

```
estudiante = {"nombre": "Ana", "edad": 25, "curso": "Python"}

# .get() - Obtener valor (no da error si no existe)
print(estudiante.get("nombre"))  # Ana
print(estudiante.get("telefono"))  # None (no existe)
print(estudiante.get("telefono", "No registrado"))  # Con valor por defecto

# .keys() - Obtener todas las claves
print(estudiante.keys())  # dict_keys(['nombre', 'edad', 'curso'])

# .values() - Obtener todos los valores
print(estudiante.values())  # dict_values(['Ana', 25, 'Python'])

# .items() - Obtener pares clave-valor
print(estudiante.items())
# dict_items([('nombre', 'Ana'), ('edad', 25), ('curso', 'Python')])
```

📖 Más Métodos de Diccionarios

```
persona = {"nombre": "Luis", "edad": 30}
# .pop() - Eliminar y retornar valor
edad = persona.pop("edad")
print(edad)  # 30
print(persona)  # {'nombre': 'Luis'}
# .update() - Actualizar/agregar múltiples pares
persona.update({"edad": 31, "ciudad": "Bogotá"})
print(persona)
# {'nombre': 'Luis', 'edad': 31, 'ciudad': 'Bogotá'}
# .clear() - Vaciar diccionario
persona.clear()
print(persona)  # {}
# in - Verificar si existe una clave
persona = {"nombre": "Ana", "edad": 25}
print("nombre" in persona)  # True
print("telefono" in persona)  # False
```

💪 Ejercicio: Trabajando con Estructuras

¡Hora de practicar! `estructuras_datos.py`

1. Crea una lista con 5 nombres de amigos e imprime el tercero
2. Agrega dos nombres más a la lista usando `.append()`
3. Crea una tupla con los días de la semana e imprime el lunes y viernes
4. Crea un diccionario con tu información personal (nombre, edad, ciudad, hobbies)
5. Imprime tu lista de hobbies del diccionario

Pistas:

- Recuerda que los índices comienzan en 0
- Las tuplas se definen con paréntesis `()`
- Los diccionarios usan llaves `{}` y formato `clave: valor`

⏱️ **Tiempo:** 15 minutos

Condicionales

Control de Flujo

¿Qué son los Condicionales?

¿Cuándo necesito esto?

Escenario: "Necesito que mi programa tome decisiones según la situación"

- **Si** hace frío → ponte abrigo
- **Si no** hace frío → usa camiseta

```
temperatura = 15

if temperatura < 18:
    print("Hace frío, ponte abrigo")
else:
    print("El clima está agradable")
```

Principio esencial: Los condicionales permiten ejecutar código solo cuando se cumple una condición.

🔀 Estructura if - Decisión Simple

```python
edad = 20

if edad >= 18:
    print("Eres mayor de edad")
    print("Puedes votar")

# El código continúa aquí sin importar la condición
print("Fin del programa")
```

Importante:

- La indentación (espacios) es OBLIGATORIA en Python
- Todo el código indentado pertenece al `if`
- Los dos puntos `:` son obligatorios

Estructura if-else - Dos Caminos

```
edad = 16

if edad >= 18:
    print("Eres mayor de edad")
    print("Puedes votar")
else:
    print("Eres menor de edad")
    print("Aún no puedes votar")

print("Gracias por participar")
```

Funcionamiento:

- Si la condición es `True` → ejecuta el bloque `if`
- Si la condición es `False` → ejecuta el bloque `else`
- Solo uno de los dos bloques se ejecuta, nunca ambos

Estructura if-elif-else - Múltiples Caminos

```
nota = 85

if nota >= 90:
    print("Calificación: A - Excelente")
elif nota >= 80:
    print("Calificación: B - Muy bien")
elif nota >= 70:
    print("Calificación: C - Bien")
elif nota >= 60:
    print("Calificación: D - Suficiente")
else:
    print("Calificación: F - Reprobado")
```

Funcionamiento:

- Evalúa condiciones en orden
- Ejecuta el primer bloque que sea `True`
- Ignora el resto aunque también sean verdaderas

⚖️ Operadores de Comparación

Para construir condiciones:

Operador	Significado	Ejemplo	Resultado
`==`	Igual a	`5 == 5`	`True`
`!=`	Diferente de	`5 != 3`	`True`
`>`	Mayor que	`8 > 5`	`True`
`<`	Menor que	`3 < 7`	`True`
`>=`	Mayor o igual	`5 >= 5`	`True`
`<=`	Menor o igual	`4 <= 3`	`False`

```
edad = 20
print(edad >= 18)  # True
print(edad == 21)  # False
```

🔗 Operadores Lógicos

Combinar múltiples condiciones:

and - Ambas condiciones deben ser verdaderas

```
edad = 25
tiene_licencia = True

if edad >= 18 and tiene_licencia:
    print("Puede conducir")
```

or - Al menos una condición debe ser verdadera

```
dia = "sábado"

if dia == "sábado" or dia == "domingo":
    print("Es fin de semana")
```

not - Invierte el valor

```
esta_lloviendo = False

if not esta_lloviendo:
    print("Puedes salir sin paraguas")
```

🔗 Ejemplos de Operadores Lógicos

```
# and - Todas deben ser True
edad = 25
tiene_dinero = True
tiene_tiempo = False

if edad >= 18 and tiene_dinero and tiene_tiempo:
    print("Puede ir al cine")  # NO se imprime
else:
    print("No puede ir")  # ✅ Se imprime
# or - Al menos una debe ser True
dia = "martes"
if dia == "sábado" or dia == "domingo" or dia == "feriado":
    print("No hay clases")
else:
    print("Hay clases")  # ✅ Se imprime

# not - Invierte el resultado
activo = False
if not activo:
    print("Usuario inactivo")  # ✅ Se imprime
```

🎯 Ejemplo Completo: Sistema de Calificaciones

```
nombre = "María"
nota = 78
asistencia = 85  # porcentaje

print(f"Evaluando a {nombre}")
if nota >= 60 and asistencia >= 80:
    if nota >= 90:
        print("Aprobado con honores")
    elif nota >= 80:
        print("Aprobado con distinción")
    else:
        print("Aprobado")
elif nota >= 60 and asistencia < 80:
    print("Aprobado pero debe mejorar asistencia")
else:
    print("Reprobado")

# Imprime: Aprobado (nota 78 ≥ 60, asistencia 85 ≥ 80)
```

💪 Ejercicio: Condicionales

Crea un archivo `condicionales.py` con:

1. Pide la edad del usuario y determina si es:
    - Menor de edad (< 18)
    - Adulto (18-65)
    - Adulto mayor (> 65)
2. Crea un programa que determine si un número es:
    - Positivo, negativo o cero
3. Crea un programa que determine si un año es bisiesto:
    - Divisible por 4 pero NO por 100, O divisible por 400

Pistas:

- Usa `input()` para recibir datos del usuario
- Convierte a `int()` los números
- Usa `elif` para múltiples opciones

⏱️ **Tiempo:** 15 minutos

Bucles

Repetir Acciones

🔄 ¿Qué son los Bucles?

¿Cuándo necesito esto?

Escenario: "Necesito repetir la misma acción muchas veces"

Analogía de la vida real:

- 🏃 Dar 10 vueltas a la pista
- 📝 Revisar cada estudiante de la lista
- 🎵 Reproducir canciones hasta que terminen

Dos tipos principales en Python:

1. **for** - Cuando sabes cuántas veces repetir
2. **while** - Cuando repites mientras se cumpla una condición

Bucles For

🔁 Bucle for con range()

Repetir un número específico de veces:

```
# range(n) - De 0 hasta n-1
for i in range(5):
    print(i)
# Imprime: 0, 1, 2, 3, 4

# range(inicio, fin) - Desde inicio hasta fin-1
for i in range(2, 6):
    print(i)
# Imprime: 2, 3, 4, 5

# range(inicio, fin, paso) - Con saltos
for i in range(0, 10, 2):
    print(i)
# Imprime: 0, 2, 4, 6, 8
```

💡 **Importante:** El número final NUNCA se incluye

🔄 Visualizando range()

Tres formas de usar range():

```
range(5)            range(2, 7)         range(0, 10, 3)
    ↓                   ↓                   ↓
[0, 1, 2, 3, 4]     [2, 3, 4, 5, 6]     [0, 3, 6, 9]
```

Ejemplos prácticos:

```python
# Imprimir números del 1 al 10
for num in range(1, 11):
    print(num)

# Cuenta regresiva
for num in range(10, 0, -1):
    print(num)
# Imprime: 10, 9, 8, 7, 6, 5, 4, 3, 2, 1

print("¡Despegue! 🚀")
```

🔄 Recorrer Cadenas con for

```python
# Iterar cada carácter de un string
palabra = "Python"

for letra in palabra:
    print(letra)
# Imprime: P, y, t, h, o, n (cada uno en una línea)

# Ejemplo práctico: contar vocales
texto = "Hola Mundo"
contador_vocales = 0

for letra in texto:
    if letra.lower() in "aeiou":
        contador_vocales += 1

print(f"Vocales encontradas: {contador_vocales}")
# Imprime: Vocales encontradas: 4
```

🔄 Recorrer Listas con for

```
# Forma 1: Iterar directamente los valores
frutas = ["manzana", "banana", "naranja", "uva"]

for fruta in frutas:
    print(f"Me gusta la {fruta}")
# Imprime:
# Me gusta la manzana
# Me gusta la banana
# Me gusta la naranja
# Me gusta la uva

# Forma 2: Iterar con índices
for i in range(len(frutas)):
    print(f"{i}: {frutas[i]}")
# Imprime:
# 0: manzana
# 1: banana
# 2: naranja
# 3: uva
```

🔄 Recorrer Diccionarios con for

```python
estudiante = {"nombre": "Ana", "edad": 22, "curso": "Python"}

# Forma 1: Iterar claves (por defecto)
for clave in estudiante:
    print(clave)
# Imprime: nombre, edad, curso

# Forma 2: Iterar valores
for valor in estudiante.values():
    print(valor)
# Imprime: Ana, 22, Python

# Forma 3: Iterar clave y valor juntos
for clave, valor in estudiante.items():
    print(f"{clave}: {valor}")
# Imprime:
# nombre: Ana
# edad: 22
# curso: Python
```

🔄 Bucles Anidados

Un bucle dentro de otro:

```
# Ejemplo: Tabla de multiplicar
for i in range(1, 4):  # Filas
    for j in range(1, 4):  # Columnas
        resultado = i * j
        print(f"{i} x {j} = {resultado}")
    print("---")  # Separador entre filas

# Imprime:
# 1 x 1 = 1
# 1 x 2 = 2
# 1 x 3 = 3
# ---
# 2 x 1 = 2
# 2 x 2 = 4
# 2 x 3 = 6
# ---
# (etc.)
```

Bucles While

🔄 Bucle while

Repetir mientras se cumpla una condición:

```
# Contar del 1 al 5
contador = 1
while contador <= 5:
    print(contador)
    contador += 1  # ¡IMPORTANTE! Actualizar la variable

# Imprime: 1, 2, 3, 4, 5

# Ejemplo práctico: pedir contraseña
contraseña_correcta = "python123"
intento = ""

while intento != contraseña_correcta:
    intento = input("Ingresa la contraseña: ")
print("¡Acceso concedido!")
```

⚠️ **Cuidado:** Si la condición siempre es `True`, ¡el bucle nunca termina!

¿Cuándo usar cada uno?

Usa `for` cuando:

- Sabes cuántas veces repetir
- Recorres una colección (lista, string, etc.)
- Trabajas con rangos numéricos

```
for i in range(10):  # Exactamente 10 veces
    print(i)
```

Usa `while` cuando:

- No sabes cuántas veces repetir
- Dependes de una condición externa
- Esperas input del usuario

```
while not encontrado:  # Hasta que se encuentre
```

⚡ break y continue

Controlar el flujo del bucle:

break - Salir del bucle inmediatamente

```
for i in range(10):
    if i == 5:
        break  # Sale del bucle cuando i es 5
    print(i)
# Imprime: 0, 1, 2, 3, 4
```

continue - Saltar a la siguiente iteración

```
for i in range(5):
    if i == 2:
        continue  # Salta el 2
    print(i)
# Imprime: 0, 1, 3, 4
```

⚡ Ejemplo: break y continue

```
# break - Buscar un número específico
numeros = [3, 7, 12, 9, 15, 4]
buscar = 9

for num in numeros:
    if num == buscar:
        print(f"¡Encontrado! {num}")
        break  # Ya lo encontré, no sigo buscando
    print(f"Revisando {num}...")
# Imprime:
# Revisando 3...
# Revisando 7...
# Revisando 12...
# ¡Encontrado! 9

# continue - Imprimir solo números pares
for num in range(10):
    if num % 2 != 0:  # Si es impar
        continue  # Saltar al siguiente
    print(num)
# Imprime: 0, 2, 4, 6, 8
```

💪 Ejercicio: Bucles Básicos

¡Hora de practicar!

Crea un archivo `bucles_practica.py` con:

1. Imprime los números del 1 al 10 usando `for`
2. Imprime los números pares del 2 al 20 usando `for` y `range()`
3. Recorre la lista `["perro", "gato", "pájaro", "pez"]` e imprime cada animal
4. Imprime los números del 5 al 1 en cuenta regresiva usando `while`
5. Suma todos los números del 1 al 100 usando un bucle

Pistas:

- Para números pares usa `range(2, 21, 2)`
- Para cuenta regresiva decrementa la variable
- Usa una variable acumuladora para sumar

⏱️ **Tiempo:** 15 minutos

🎯 Ejercicio CORE: Bucles for Básico 1

Desafío para practicar en casa:

Crea `bucle_for_basico1.py` con estas funciones:

1. **Básico**: Imprime del 0 al 100
2. **Múltiples de 2**: Imprime múltiplos de 2 entre 2 y 500
3. **Ice Ice Baby**: Del 1 al 100, si es divisible por 5 imprime "ice ice", si es por 10 imprime "baby"
4. **Número gigante**: Suma números pares del 0 al 500,000
5. **Cuenta regresiva**: De 2024 hacia atrás de 3 en 3
6. **Contador dinámico**: Variables inicio, fin y múltiplo, imprime múltiplos en ese rango

💡 **Tiempo recomendado:** 30-45 minutos

💡 **Este ejercicio es CORE - obligatorio completarlo**

Resumen y Cierre

📝 Conceptos Clave de Hoy

Lo más importante:

✅ **Variables** - Cajas para guardar información

✅ **Tipos de datos** - Boolean, Integer, Float, String

✅ **Listas** - Colecciones mutables y ordenadas `[]`

✅ **Tuplas** - Colecciones inmutables `()`

✅ **Diccionarios** - Pares clave-valor `{}`

✅ **Condicionales** - `if`, `elif`, `else` para tomar decisiones

✅ **Bucles** - `for` para iterar, `while` para repetir condicionalmente

🎯 Lo Más Importante

Recuerda:

- 💡 **Python es sensible a mayúsculas** - `nombre` ≠ `Nombre`
- 💡 **La indentación es obligatoria** - Define bloques de código
- 💡 **Los índices comienzan en 0** - Primer elemento es `[0]`
- 💡 **Las listas son mutables, las tuplas no** - Elige según necesites
- 💡 **Los diccionarios son ideales para datos relacionados** - clave: valor
- 💡 `for` **cuando sabes cuánto repetir,** `while` **cuando no** - Cada uno tiene su uso

📚 Recursos para Seguir Aprendiendo

Documentación y práctica:

- 📖 Documentación Oficial de Python - Referencia completa
- 🎮 Python Tutor - Visualiza cómo se ejecuta tu código
- 💻 HackerRank Python - Ejercicios prácticos
- 📚 Real Python - Tutoriales de calidad
- 🎯 Codewars - Desafíos de programación
- 📝 W3Schools Python - Tutoriales interactivos

🚀 Próximos Pasos

¿Qué viene después?

En las próximas clases veremos:

- 🎭 **Funciones** - Reutilizar código de forma eficiente
- 📦 **POO (Programación Orientada a Objetos)** - Clases y objetos
- 📁 **Archivos** - Leer y escribir archivos
- 🌐 **Flask** - Crear aplicaciones web con Python
- 🗃️ **Bases de datos** - Conectar Python con MySQL

¡Practicar los bucles y estructuras de datos es esencial para todo lo que viene!

Ejercicios recomendados:

1. **FizzBuzz Clásico**: Del 1 al 100
    - Divisible por 3: "Fizz"
    - Divisible por 5: "Buzz"
    - Divisible por ambos: "FizzBuzz"
2. **Lista de Compras**: Crea un programa que:
    - Permita agregar productos a una lista
    - Muestre todos los productos
    - Permita eliminar productos
3. **Agenda de Contactos**: Usa un diccionario para:
    - Guardar nombre y teléfono
    - Buscar contactos

💡 Consejos Finales

Buenas prácticas:

✅ **Usa nombres descriptivos** - `edad_usuario` en vez de `x`

✅ **Comenta tu código** - Explica el "por qué", no el "qué"

✅ **Practica todos los días** - Aunque sea 30 minutos

✅ **Lee código de otros** - Aprende diferentes enfoques

✅ **No memorices, entiende** - Los detalles los buscas después

✅ **Usa print() para depurar** - Ver qué hacen tus variables

✅ **Pide ayuda cuando te atores** - La comunidad Python es muy amigable

¡No cometas estos errores!

❌ Olvidar los dos puntos `:` después de `if`, `for`, `while`

```
if edad > 18  # ❌ Falta :
```

❌ Mezclar indentación (tabs y espacios)

```
if True:
    	print("Mal")  # Tab
  print("indentado")  # Espacios
```

❌ Intentar modificar tuplas

```
tupla = (1, 2, 3)
tupla[0] = 5  # ❌ Error!
```

❌ Olvidar convertir tipos

```
edad = input("Edad: ")  # Es string!
```

🎉 ¡Felicidades!

Ya conoces los fundamentos de Python

Ahora puedes crear programas que toman decisiones, procesan datos y automatizan tareas 🚀

❓ Preguntas

¿Alguna duda sobre los conceptos de hoy?

Recuerda: No hay preguntas tontas, solo oportunidades de aprendizaje 💭

🙌 ¡Excelente Trabajo!

¡Nos vemos en la siguiente clase con Funciones en Python!

No olvides completar el ejercicio CORE de Bucles for Básico 1 📝

¡Sigue practicando y nos vemos pronto! 🐍✨