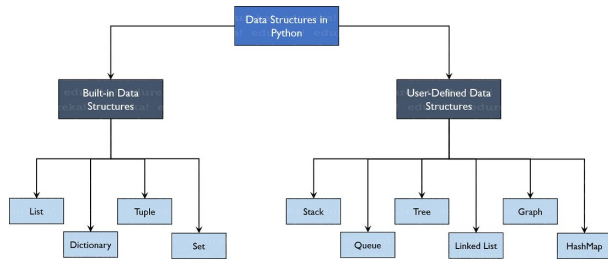


Estructuras de datos y de control en Python

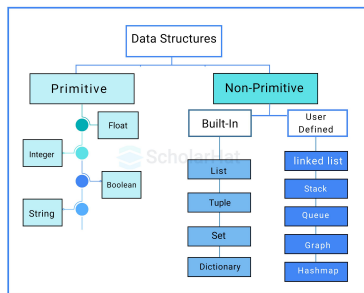
Estructuras de Datos y de Control en Python

Fundamentos esenciales para la programación eficiente

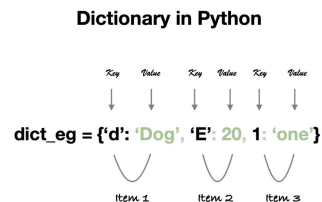
Las estructuras de datos y de control son los pilares fundamentales de la programación en Python. Las estructuras de datos nos permiten organizar y manipular información de manera eficiente, mientras que las estructuras de control determinan el flujo y la lógica de nuestros programas. Dominar estos conceptos es esencial para desarrollar aplicaciones robustas y eficientes en Python.



Estructuras de Datos en Python



	Mutable	Ordered	Indexing / Slicing	Duplicate Elements
List	✓	✓	✓	✓
Tuple	✗	✓	✓	✓
Set	✓	✗	✗	✗



Listas

- Ordenadas y mutables
- Permiten elementos duplicados y diferentes tipos
- Acceso por índice y métodos como `append()`

RAFAEL MARTÍN R.

Tuplas y Sets

- Tuplas: inmutables y ordenadas
- Sets: elementos únicos y desordenados
- Operaciones matemáticas en sets

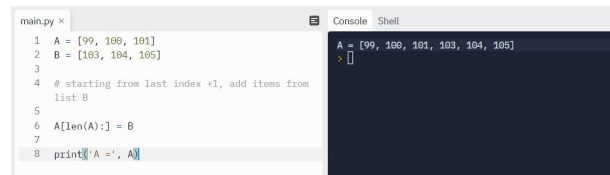
Diccionarios

- Estructura de pares clave-valor
- Acceso rápido mediante claves únicas
- Valores pueden ser de cualquier tipo

Listas en Python

Estructura de datos ordenada y mutable

- Creación: `lista = [1, 2, 3]` o `lista = list()`
- Acceso por índice: `lista[0]` para primer elemento
- Método `append()` añade elementos al final
- Método `pop()` elimina y retorna el último elemento
- Método `extend()` combina dos listas
- Modificación directa: `lista[1] = nuevo_valor`

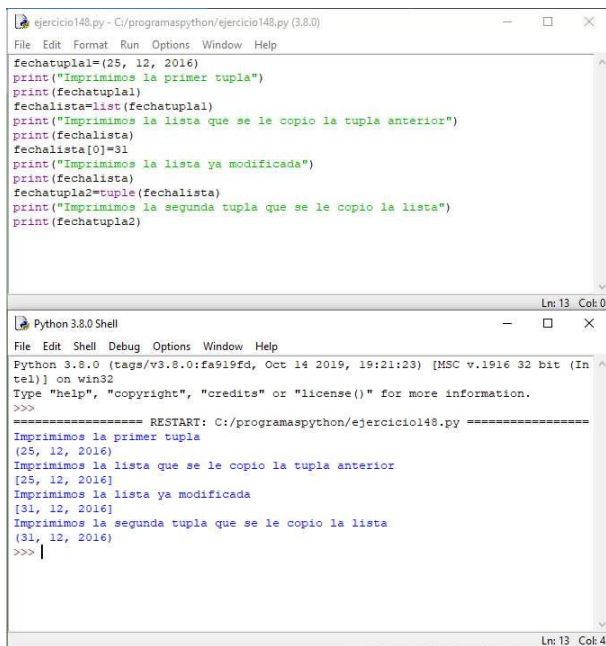


The screenshot shows a Python IDE with a file named `main.py` and a console window. The code in `main.py` defines two lists, `A` and `B`, and then appends the elements of `B` to `A` using `A[len(A):] = B`. The console output shows the final state of list `A` as `[99, 100, 101, 103, 104, 105]`.

```
main.py x Console Shell
1 A = [99, 100, 101]
2 B = [103, 104, 105]
3
4 # starting from last index +1, add items from
  list B
5
6 A[len(A):] = B
7
8 print('A =', A)
```

```
A = [99, 100, 101, 103, 104, 105]
```

Tuplas en Python



The image shows a screenshot of a Python IDE with two windows. The top window is a text editor titled 'ejercicio148.py' containing the following Python code:

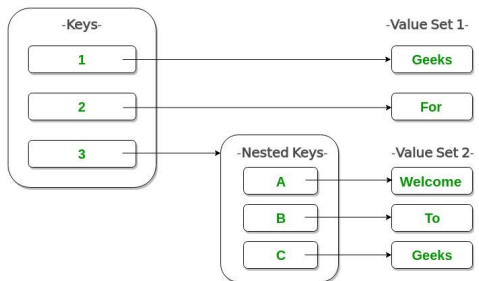
```
fechatupla1=(25, 12, 2016)
print("Imprimimos la primer tupla")
print(fechatupla1)
fechalista=list(fechatupla1)
print("Imprimimos la lista que se le copio la tupla anterior")
print(fechalista)
fechalista[0]=31
print("Imprimimos la lista ya modificada")
print(fechalista)
fechatupla2=tuple(fechalista)
print("Imprimimos la segunda tupla que se le copio la lista")
print(fechatupla2)
```

The bottom window is a 'Python 3.8.0 Shell' showing the execution output of the code above:

```
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/programasp/python/ejercicio148.py =====
Imprimimos la primer tupla
(25, 12, 2016)
Imprimimos la lista que se le copio la tupla anterior
[25, 12, 2016]
Imprimimos la lista ya modificada
[31, 12, 2016]
Imprimimos la segunda tupla que se le copio la lista
(31, 12, 2016)
>>> |
```

Las tuplas son estructuras de datos inmutables en Python, lo que significa que una vez creadas, sus elementos no pueden modificarse. Se definen usando paréntesis () y son ideales para almacenar datos que no deben cambiar, como coordenadas geográficas o información de configuración. A diferencia de las listas, las tuplas son más eficientes en memoria y pueden usarse como claves en diccionarios.

Diccionarios en Python



Estructura y Características

- Colección de pares clave-valor usando {}
- Claves deben ser únicas e inmutables
- Acceso rápido mediante claves específicas



Métodos Principales

- `dict.keys()` devuelve todas las claves
- `dict.values()` obtiene todos los valores
- `dict.items()` retorna pares clave-valor

Sets en Python

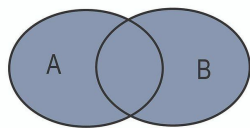
Set Operations and Venn Diagrams

example
sets

$A = \{1,2,3,4\}$ $B = \{3,4,5,6,7\}$

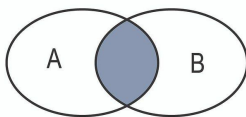
Union

$A \cup B = \{1,2,3,4,5,6,7\}$

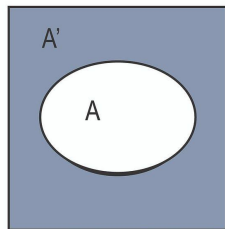


Intersection

$A \cap B = \{3,4\}$

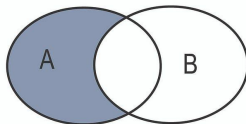


Complement
of A



Difference

$A - B = \{1,2\}$



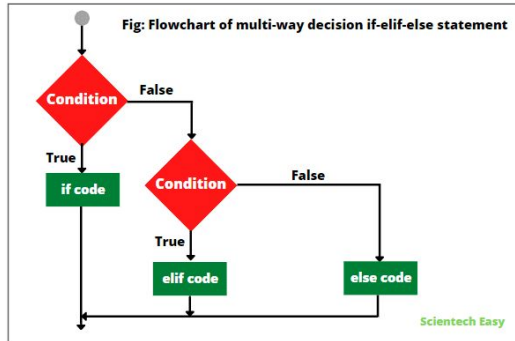
- Sets son colecciones desordenadas de elementos únicos
- Creación usando `set()` o llaves `{}`: `mi_set = {1, 2, 3}`
- Operación `union()`: combina elementos de dos sets
- Operación `intersection()`: encuentra elementos comunes entre sets
- Operación `difference()`: obtiene elementos únicos del primer set
- Métodos `add()` y `remove()` para modificar elementos

Estructuras de Control: Introducción

Herramientas fundamentales para dirigir el flujo del programa

Las estructuras de control son elementos fundamentales en Python que permiten controlar el flujo de ejecución de un programa. Estas herramientas nos permiten tomar decisiones (estructuras condicionales) y repetir acciones (bucles) según las necesidades específicas de nuestro código. Su dominio es esencial para crear programas eficientes y funcionales, ya que nos permiten automatizar tareas repetitivas y manejar diferentes escenarios en nuestras aplicaciones.

Estructuras Condicionales: if, elif, else



```
File Edit Format Run Options Window Help
puntuacion=float(input('Introduce tu puntuacion: '))

if puntuacion>=90:
    print('Grado = "A"')
elif puntuacion>=80:
    print('Grado = "B"')
elif puntuacion>=70:
    print('Grado = "C"')
elif puntuacion<70:
    print('Grado = "D"')
else:
    print('Grado = "F"')

print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
```

Grado 'A' → mayores o iguales que 90 (≥90)
Grado 'B' → menores que 90 y mayores o iguales que 80 (≥80)
Grado 'C' → menores que 80 y mayores o iguales que 70 (≥70)
Grado 'D' → menores que 70 y mayores o iguales que 60 (≥60)
Grado 'F' → menores de 60 (<60)

Sintaxis y Flujo

Ejemplo Práctico

Bucle for en Python

Herramienta poderosa para iterar secuencias de datos

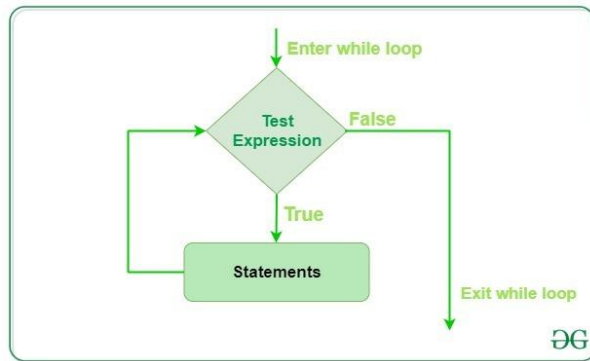
- Sintaxis básica: for elemento in secuencia
- Iteración con range() para control numérico
- Recorrido de listas, tuplas y diccionarios
- Break para salir del bucle completamente
- Continue para saltar a la siguiente iteración
- Anidamiento de bucles for para matrices

```
animals = ["cat", "dog", "penguin", "whale", "zebra"]  
  
for x in animals:  
    print(x)  
  
# You could read this as "for every animal 'x' in the list 'animals', print the animal 'x'."  
# 'x' is known as the iterator and 'animals' is known as the iterable  
  
cat  
dog  
penguin  
whale  
zebra
```

Bucle while en Python

Control de flujo con iteración condicional

El bucle while en Python ejecuta un bloque de código mientras una condición específica sea verdadera. Es ideal para situaciones donde no conocemos el número exacto de iteraciones necesarias. La estructura básica incluye una condición de entrada, el bloque de código a ejecutar, y generalmente una actualización de variables para evitar bucles infinitos. Se puede controlar su flujo usando `break` para salir del bucle o `continue` para saltar a la siguiente iteración.



Comprensión de Listas

Ventajas de List Comprehension

- Código más conciso y legible
- Mayor rendimiento que bucles tradicionales
- Sintaxis elegante y pythónica
- Facilita operaciones de mapeo y filtrado

Consideraciones de Uso

- Puede ser confuso para principiantes
- No recomendado para lógica muy compleja
- Puede afectar legibilidad si se anida

Manejo de Excepciones

Control de errores con try-except en Python

```
def add_numbers():
    num1 = input()
    num2 = input()
    result = 0

    try:
        result = int(num1) + int(num2)
    except ValueError:
        print('there was an error')
        raise
    else:
        return result

add_numbers()
10
20
30
add_numbers()
1
a
there was an error
Traceback (most recent call last):
  File "<pyshell#111>", line 1, in <module>
    add_numbers()
  File "<pyshell#109>", line 7, in add_numbers
    result = int(num1) + int(num2)
ValueError: invalid literal for int() with base 10: 'a'
```

- Try: Bloque de código que puede generar una excepción
- Except: Captura y maneja errores específicos como ValueError
- Finally: Ejecuta código independientemente de las excepciones
- Raise: Permite lanzar excepciones de forma manual
- Except múltiple: Maneja diferentes tipos de errores
- Else: Se ejecuta cuando no hay excepciones

Mejores Prácticas

Optimiza tu código Python con estas recomendaciones clave

- Elige la estructura de datos adecuada según los requisitos
- Utiliza list comprehension para código más limpio y eficiente
- Prefiere for sobre while cuando conozcas el número de iteraciones
- Implementa manejo de excepciones en operaciones críticas
- Evita bucles anidados profundos para mejor rendimiento
- Usa diccionarios para búsquedas frecuentes de datos

Ejercicios Prácticos

Ejercicios Propuestos

- Crear un diccionario de estudiantes usando list comprehension
- Implementar búsqueda en lista usando while y try-except
- Desarrollar un programa de gestión con sets

Soluciones Clave

- Usar dict() con zip() para mapear nombres y notas
- Implementar break cuando se encuentra el elemento
- Aplicar operaciones de conjuntos para validar datos

Resumen y Recursos Adicionales

Recursos clave para dominar Python

Python ofrece una rica variedad de estructuras de datos y control que permiten escribir código eficiente y elegante. Para profundizar tus conocimientos, consulta la documentación oficial de Python, plataformas como Real Python y Python.org, y practica en sitios como CodeAcademy o LeetCode. Recuerda que la práctica constante es clave para dominar estos conceptos fundamentales.

