UNIVERSIDAD TECNOLOGICA NACIONAL

Capítulo 2: Variables y Colecciones

Programación 2

Asignación de variables

No hay necesidad de declarar las variables antes de asignarlas

```
una_variable = 5        # La convención es usar
                          guiones_bajos_con_minúsculas

una_variable: float = 5  # Type-Hints opcionales |
                                RECOMENDADO

una_variable          # => 5

otra_variable         # Error variable no asignada
```

Operadores con reasignación y asignación múltiple

```
una_variable                    # => 5

una_variable += 2               # una_variable == 7

una_variable -= 1               # una_variable == 6

una_variable *= 5               # una_variable == 30

una_variable /= 5               # una_variable == 3.0

una_variable **= 3              # una_variable ==
27.0

una_variable %= 10              # una_variable == 7.0

una_variable //= 5              # una_variable == 1.0
```

Asignación múltiple

```
a = b = 3                         # a = 3 y b = 3
```

Colecciones: listas

Las listas no tienen longitud fija ni tipo de dato predeterminado.

```
lista = []                    # Vacia

otra = [4, 5, 6]              # Con valores por defecto

multiple = [2, "Juan", [2]]   # Valores de distinto tipo
```

Colecciones: listas

Existen diferentes métodos que pueden ser aplicados a listas.

```
lista.append(1)          # Agregar un elemento al final
lista.extend([2, 4, 3])  # Agregar múltiples elementos
lista.pop()              # => 3 y lista=[1, 2, 4]
lista.insert(3, 3)       # Agregar un elemento en la posición dada
len(lista)               # 4
```

Colecciones: listas

Indexación simple y múltiple en listas:

Indexado Simple

```
lista[0]      # => 1 Primer elemento
lista[-1]     # => 4 Último elemento
lista[4]      # Error - Fuera de los límites
```

Indexado Múltiple lista [inicio:final:pasos]

lista = [1,2,3,4]

```
lista[1:3]    # => [2, 3] (del segundo elemento al cuarto sin incluir)
lista[2:]     # => [3, 4] (del tercer elemento en adelante)
lista[:3]     # => [1, 2, 3] (todo hasta el cuarto elemento sin incluir)
lista[::2]    # => [1, 3] (todos los elementos pero con pasos de 2)
lista[::-1]   # => [4, 3, 2, 1] (todos los elementos invertidos)
a = lista[:]      # => Crea una copia idéntica de lista
a = lista             # => a es igual a lista. Si cambio lista, cambia a.
```

Colecciones: listas

Operaciones

Operaciones con Listas

```
lista + otra          # => [1, 2, 4, 3, 4, 5, 6]
lista * 2             # => [1, 2, 4, 3, 1, 2, 4, 3]
```

Operador in

```
1 in lista            # => True
not 5 in lista        # => True El operador not puede estar antes o
                           después
5 not in lista        # => True
```

Operador ==

```
lista == [1, 2, 4, 3]     # => True
lista == lista[:]         # => True
lista == lista            # => True
```

Operador is

```
lista is [1, 2, 4, 3]     # => False
lista is lista[:]         # => False
lista is lista            # => True
```

Operaciones especiales para listas booleanas

```
any(lista)            # => True | Devuelve True si al menos uno de los
                           elementos es True
all(lista)            # => True | Devuelve True si todos los
                           elementos son True
```

Colecciones: tuplas

Es un tipo de dato que se usa mayormente de manera implícita. La diferencia entre una tupla y una lista es que la tupla es inmutable. Esto quiere decir que una vez definida la tupla no puede modificarse.

```
tupla = (1, 2, 3)  # Se definen con (,) en lugar de []
tupla = 1, 2, 3    # Los paréntesis son opcionales
tupla[0]           # => 1
tupla[0] = 3       # TypeError
```

Métodos idénticos a las listas pero sin asignación

```
len(tupla)         # => 3
tupla + (4, 5, 6)  # => (1, 2, 3, 4, 5, 6)
tupla[:2]          # => (1, 2)
2 in tupla         # => True
```

Colecciones: tuplas

Desempaquetado

Desempaquetado Simple

```
a, b, c = (1, 2, 3)          # a == 1, b == 2, c == 3
a, b, c = [1, 2, 3]          # a == 1, b == 2, c == 3
a, b = [1, 2, 3]             # Error | Cantidad de elementos
                               debe ser idéntica
a, b = b, a                  # Intercambio a == 2, b == 1
```

Desempaquetado Con comodines

```
a, *rest = [1, 2, 3, 4]      # a == 1, rest == [2, 3, 4]
*rest, b = [1, 2, 3, 4]      # b == 4, rest == [2, 3, 4]
a, *rest, b = [1, 2, 3, 4]   # a == 1, b == 4, rest == [2, 3]
```

Desempaquetado Anidado

```
(a, b), c = [[1, 2], [3]]    # a == 1, b == 2, c == [3]
```

Colecciones: Diccionarios

Una de las ventajas principales de los diccionarios es que el tiempo de acceso es mucho más rápido que una lista. Sin embargo, es necesario tener una clave con la cual se acceden a los valores y el valor propiamente dicho.

```
diccionario_vacio = {}          # Vacio

diccionario = {"uno": 1,        # Declaración multilinea
               "dos": 2,
               "tres": 3,       # Coma al final válida
               }
diccionario["uno"]              # => 1 - Indexado con Claves

diccionario["cuatro"]           # Error


diccionario.get("uno")          # => 1

diccionario.get("cuatro")       # => None en vez de Error

diccionario.get("uno", 4)       # => 1

diccionario.get("cuatro", 4)    # => Valor por defecto en lugar de None
```

Colecciones: Diccionarios

Métodos y operadores:

Métodos

```
list(diccionario.keys())      # => ["tres", "dos", "uno"]

list(diccionario.values())    # => [3, 2, 1]

list(diccionario.items())     # => [('uno', 1), ('dos', 2), ('tres', 3)]
```

Operadores con Diccionarios | in verifica las claves

```
"uno" in diccionario          # => True

1 in diccionario              # => False
```

Colecciones: Diccionarios

Actualización de diccionarios

```
nuevos_datos = {"cuatro": None, "cinco": 5}

diccionario.update(nuevos_datos)

diccionario  # {'uno': 1, 'dos': 2, 'tres': 3, 'cuatro': None, 'cinco': 5}
```

Colecciones: conjuntos

Los conjuntos suelen ser útiles para filtrar elementos repetidos en una colección. Además permiten las operaciones tradicionales de conjuntos que pueden ser útiles en algunos contextos.

```
conjunto_vacio = set()

conjunto = {1, 2, 2, 3, 4}

conjunto.add(5)  # conjunto ahora es {1, 2, 3, 4, 5}
```

Error común:

```
conjunto = {}          #Se está inicializando un
                         diccionario y no un conjunto.
```

Colecciones: conjuntos

Operaciones:

conjunto = {1, 2, 2, 3, 4}

otro_conjunto = {3, 4, 5, 6}

```
conjunto & otro_conjunto                    # => {3, 4, 5} | Intersección

conjunto.intersection(otro_conjunto)        # => {3, 4, 5} | Intersección

conjunto | otro_conjunto                    # => {1, 2, 3, 4, 5, 6} | Union

conjunto.union(otro_conjunto)               # => {1, 2, 3, 4, 5, 6} | Union

conjunto - otro_conjunto                    # => {1, 2} | Diferencia

conjunto.difference(otro_conjunto)          # => {1, 2} | Diferencia

conjunto ^ otro_conjunto                    # => {1, 2, 5, 6} | Diferencia
                                                simétrica

conjunto.symmetric_difference(otro_conjunto)  # => {1, 2, 5, 6} |
                                                Diferencia Simétrica
```

Colecciones: conjuntos

Métodos

```
conjunto.isdisjoint(otro_conjunto)    # => False

conjunto.issubset(otro_conjunto)      # => False

conjunto.issuperset(otro_conjunto)    # => False
```