Introducción a la Programación

Prof. Agustín Gravano

Primer semestre de 2022

Clase teórica 19: Tuplas, conjuntos y diccionarios

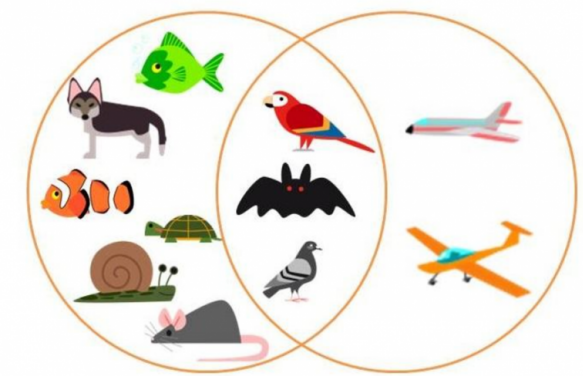
Tipos de datos

Hasta ahora vimos tipos: `bool`, `int`, `float`, `str`, `List[T]`.

Los tipos de datos son útiles para representar datos, y también para escribir algoritmos más simples.

Esta semana vamos a ver 3 tipos compuestos:

- tuplas (dupla, tripla, cuádrupla, quíntupla, etc.)
- conjuntos



- diccionarios: clave $\rightarrow$ valor

Tipos de datos · Tupla

Las tuplas se usan para representar y almacenar múltiples valores en una única variable. Por ejemplo:

```
punto:Tuple[float, float, float] = (1.7, -0.9, 0.4)
```

`(1.7, -0.9, 0.4)` es una tupla formada por tres floats, y representa un punto en el espacio tridimensional.

En general:

- Los valores van entre paréntesis y separados por comas: `(1, 2, 3)`
- Los valores pueden ser de distintos tipos: `(0.1, 'hola', False)`
- Las tuplas son inmutables: una vez creadas, no se pueden agregar, modificar o borrar sus valores.

Ejemplos:

```
fecha:Tuple[int, str, int] = (25, 'mayo', 1810)
persona:Tuple[str, str, int] = ('Juana', 'Fulánez', 27)
```

En Python, para usar tuplas en las sugerencias de tipos (*type hints*), debemos incluir `from typing import Tuple`.

Tipo Tupla (`Tuple[T0, T1, ...]`) · Operaciones

Dadas `t`, `u` de tipo `Tuple[T0, T1, T2]`; `x`, `y`, `z` de tipos `T0`, `T1`, `T2`, respectivamente:

- `(x, y, z)` Crea y devuelve una tupla con tres valores.
- `len(t)` Devuelve la cantidad de componentes de la tupla `t`.
- `t[i]` Devuelve el valor de la `i`-ésima componente de `t` (Requiere: $0 \leq$ `i` $<$ `len(t)`)
- `t == u` Compara `t` y `u`, componente a componente.

La asignación de tuplas permite hacer asignaciones simultáneas:

```
a:str = 'ritmo'
b:str = 'algo'
print(a + b)          # imprime 'ritmoalgo'

(a, b) = (b, a)       # asignación de tuplas
print(a + b)          # imprime 'algoritmo'
```

Tipo Tupla (`Tuple[T0, T1, ...]`)

Por favor, ¡no confundir tuplas y listas!

Una tupla `t` es solo una colección fija (inmutable) de valores.

No podemos modificar una tupla:

- ~~`t.append(valor)`~~
- ~~`t[i] = valor`~~
- ~~`t.pop()`~~
- etc.

Lo más sencillo es pensar a una tupla como un par ordenado (una *dupla*), o una terna ordenada (una *tripla*), etc.

Tipos de datos · Conjunto

Python provee un tipo `Set[T]` para representar conjuntos. Por ejemplo:

```
felinos:Set[str] = {'león', 'gato', 'tigre', 'guepardo'}
```

En general:

- Los elementos van entre llaves, separados por comas y (en esta materia) son todos del mismo tipo:

```
{0.1, 3.1415, math.sqrt(2)}
```

- A diferencia de las listas, en los conjuntos no hay elementos repetidos, y tampoco aparecen en un cierto orden.
- Los conjuntos son mutables: se pueden agregar y borrar elementos durante la ejecución de un programa.

En Python, para usar conjuntos en las sugerencias de tipos (*type hints*), debemos incluir `from typing import Set`.

Tipo Conjunto (`Set[T]`) · Operaciones

Dadas las variables `c,d` de tipo `Set[T]`; `x`, `y`, `z` de tipo `T`; `ls` de tipo `List[T]`:

- `set()` — Crea y devuelve un conjunto vacío.
- `{x, y, z}` — Crea y devuelve un conjunto con 3 elementos.
- `len(c)` — Devuelve la cantidad de elementos de `c`.
- `x in c` — Dice si el elemento `x` está en el conjunto `c`.
- `c.add(x)` — Agrega el elemento `x` al conjunto `c`.
- `c.remove(x)` — Elimina el elemento `x` de `c`. (Requiere: `x in c`)
- `c == d` — Dice si los dos conjuntos son iguales (devuelve un `bool`).
- `c | d` — Devuelve un nuevo conjunto con $c \cup d$ (unión).
- `c & d` — Devuelve un nuevo conjunto con $c \cap d$ (intersección).
- `c - d` — Devuelve un nuevo conjunto con $c \setminus d$ (diferencia).
- `list(c)` — Devuelve una lista nueva con los elementos del conjunto `c` (en algún orden arbitrario).
- `set(ls)` — Devuelve un conjunto nuevo con los elementos de la lista `ls` (eliminando los duplicados).

Tipo Conjunto (`Set[T]`) · Ejemplos

```python
from typing import Set

felinos1:Set[str] = set()
felinos1.add('león')
felinos1.add('gato')

felinos2:Set[str] = set()
felinos2.add('gato')
felinos2.add('león')
felinos2.add('león')

felinos1 == felinos2        # Devuelve True

felinos1.add('tigre')
felinos2.add('puma')
felinos1 & felinos2
    # Intersección. Devuelve {'gato', ' león'}
felinos1 | felinos2
    # Unión. Devuelve {'puma', 'gato', ' león', 'tigre'}
felinos1 - felinos2
    # Diferencia. Devuelve {'tigre'}
```

Tipo Conjunto (`Set[T]`) · Ejercicios

Programar las siguientes funciones:

```
1  def caracteres(s:str) -> Set[str]:
2      ''' Requiere: nada.
3          Devuelve: el conjunto de los caracteres que aparecen en s.
4          Ejemplo: caracteres('bananas') → {'b', 'a', 'n', 's'}
5      '''
6
7  def caracteres_en_comun(s:str, t:str) -> Set[str]:
8      ''' Requiere: nada.
9          Devuelve: el conjunto de los caracteres que aparecen
10         tanto en s como en t.
11         Ej: caracteres_en_comun('bananas', 'peras') → {'a', 's'}
12     '''
```

Operaciones de conjuntos:

`set()`:	nuevo conjunto	`x in c`:	pertenencia	`c == d`	
`{x,y,z}`:	nuevo conjunto	`c.add(x)`:	agregar	`c & d`:	intersección
`set(ls)`:	nuevo conjunto	`c.remove(x)`:	eliminar	`c | d`:	unión
`len(c)`:	cantidad	`list(c)`		`c - d`:	diferencia

Tipo Diccionario (`Dict[T1, T2]`)

Un diccionario asocia valores (de algún tipo) a claves (de algún tipo).

20

A

a primera letra del abecedario; primera vocal; preposición que precede a determinados complementos verbales.
ábaco m. instrumento para contar; parte superior del capitel.
abad m. superior de una abadía; prior; rector.
abadía f. monasterio.
abajo ad. en un lugar que está más bajo; debajo.
abalorio m. cuentecilla de vidrio con abertura redonda; lentejuela.
abanderado m. el que porta la bandera.
abandonar tr. r. apartarse de alguna persona, cosa o asunto.
abanicar tr. r. hacer aire con el abanico.
abanico m. instrumento que sirve para hacer aire.
abaratar tr. r. reducir el precio; hacer algo más barato.
abarcar tr. rodear; comprender; hacerse cargo de varios asuntos a un tiempo; englobar.
abastecer tr. suministrar provisiones; avituallar.
abatimiento m. decaimiento físico o moral.
abatir tr. derribar; derrocar; decaer el ánimo; humillar.
abdicar tr. renunciar o ceder algún derecho, facultad o poder.
abdomen m. cavidad del cuerpo humano que contiene

País	Población
China	1 410 183 000
India	1 389 754 000
Estados Unidos	334 398 000
Indonesia	272 616 000
Pakistán	225 200 000
Brasil	212 897 000

Teléfonos útiles

911 Policía
100 Bomberos
103 Emergencias
107 SAME
108 Atención Social
147 Atención Ciudadana

Tipo Diccionario (`Dict[T1, T2]`) · Operaciones

Dadas las variables `d` de tipo `Dict[T1, T2]`; `k` de tipo `T1`; `v` de tipo `T2`:

- `dict()` — Devuelve un diccionario nuevo y vacío.
- `{}` — Devuelve un diccionario nuevo y vacío.
- `len(d)` — Devuelve la cantidad de claves del diccionario `d`.
- `k in d` — Dice si la clave `k` está definida en el diccionario `d`.
- `d[k] = v` — Asocia el valor `v` a la clave `k` en el diccionario `d`.
- `d[k]` — Devuelve el valor asociado a la clave `k` en el diccionario `d` (Requiere: `k in d`)
- `d.pop(k)` — Borra la clave `k` y su valor asociado. (Requiere: `k in d`)
- `list(d)` — Devuelve una lista con las claves del diccionario `d` (en algún orden arbitrario).

En Python, para usar diccionarios en las sugerencias de tipos (*type hints*), debemos incluir `from typing import Dict`.

Al igual que las listas y los conjuntos, los diccionarios son mutables.

Tipo Diccionario (`Dict[T1, T2]`) · Ejemplo

```python
from typing import Dict

peso_atómico:Dict[str, float] = dict()
len(peso_atómico)       # devuelve 0

peso_atómico['H']  = 1.00797
peso_atómico['He'] = 4.0026
peso_atómico['C']  = 12.0111
peso_atómico['U']  = 238.02891

'Ag' in peso_atómico    # devuelve False
peso_atómico['Ag'] = 107.8683
'Ag' in peso_atómico    # devuelve True

len(peso_atómico)       # devuelve 5
list(peso_atómico)      # devuelve ['H','He','C','U','Ag']
```

Tipo Diccionario (`Dict[T1, T2]`) · Ejercicio:

Programar la siguiente función:

```
1  def recitar(n:int) -> str:
2      ''' Requiere: n >= 0
3          Devuelve: un string con la recitación de los dígitos de n,
4                    en minúsculas y separados por espacios en blanco.
5          Ejemplos: recitar(1234) --> 'uno dos tres cuatro'
6                    recitar(0)-->'cero'
7                    recitar(999)-->'nueve nueve nueve'
8      '''
```

Operaciones de diccionarios:

`dict()`:	nuevo diccionario	`d[k]`:	consultar
`{}`:	nuevo diccionario	`d(k) = v`:	asociar
`len(d)`:	cantidad de claves	`d.pop(k)`:	eliminar
`k in d`:	existencia de clave	`list(d)`:	claves

Iteradores para conjuntos y diccionarios

La instrucción for nos permite iterar sobre los elementos de un conjunto:

```
felinos:Set[str]  = {'león', 'gato', 'tigre', 'puma'}
for f in felinos:
    print(f)
```

for también nos permite iterar sobre las claves de un diccionario:

```
peso_atómico:Dict[str,float] = {'H': 1.00797,
                                'He': 4.0026,
                                'C': 12.0111,
                                'U': 238.02891}
for elemento in peso_atómico:
    print('El peso atómico del', elemento,
          'es', peso_atómico[elemento])
```

Complejidad algorítmica

- Al igual que con listas, averiguar siempre la complejidad algorítmica de las operaciones de conjuntos y diccionarios.
- `https://wiki.python.org/moin/TimeComplexity`

Testing de unidad

- La biblioteca `unittest` ofrece dos operaciones muy útiles para listas, conjuntos y diccionarios:
  - `assertIn(v, c)`: Revisa que `v` esté definido en `c`.
  - `assertNotIn(v, c)`: Revisa que `v` no esté definido en `c`.

  donde `c` es un contenedor (`List[T]`, `Set[T]` o `Dict[T, W]`) y `v` es de tipo `T`.

Repaso de la clase de hoy

- Tuplas, conjuntos.
- Operaciones, mutabilidad, operadores.

Bibliografía complementaria:

- APPP2, secciones 9.1 a 9.3; capítulo 10 (excepto 10.6, que está deprecado en Python3).
- HTCSP3, capítulo 9, capítulo 20 (excepto 20.5).

Con lo visto, pueden resolver la Guía de Ejercicios 8 completa.