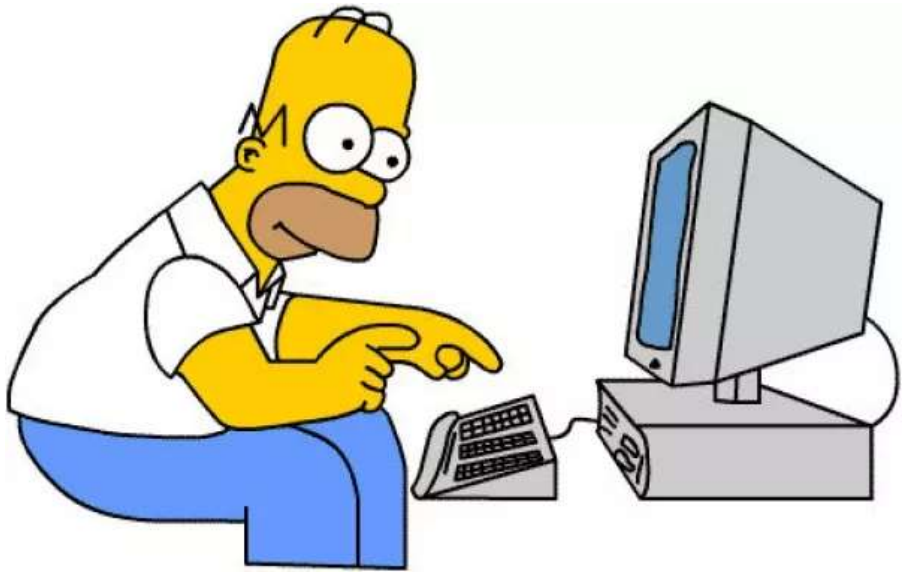
INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.



INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

En informática, una **estructura de datos** es una forma de organizar, gestionar y almacenar conjuntos de datos para acceder a ellos y manipularlos de manera eficiente, de acuerdo con el problema que estamos resolviendo.

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Estructura de datos.

En general, la información tratada por el ordenador irá agrupada de una forma más o menos coherente en estructuras especiales, compuestas por datos simples. A este tipo de agrupaciones las denominaremos **tipos de datos estructurados o tipos compuestos**.

Los tipos de datos estructurados o tipos compuestos se distinguen por los elementos que las componen y por las operaciones que se pueden realizar con ellas o entre sus elementos.

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Estructura de datos.

Una “estructura de datos” es una **colección de valores**, **la relación que existe entre estos valores y las operaciones que podemos hacer sobre ellos**; en pocas palabras se refiere a **cómo los datos están organizados y cómo se pueden administrar**. Una estructura de datos describe el formato en que los valores van a ser almacenados, cómo van a ser accedidos y modificados, pudiendo así existir una gran cantidad de estructuras de datos.

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Estructura de datos.

Las estructuras de datos:

- **Establecen una organización de los elementos almacenados.** Podemos representarlas bajo una única variable. Dicha variable representa así la colección de valores.
- **Proporcionan un conjunto de operaciones para manipular estos elementos.** Nos facilitan acciones como recuperar, borrar, insertar o buscar elementos; en definitiva, trabajar con la estructura.

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Estructura de datos - Arreglos, matrices, vectores o arrays.

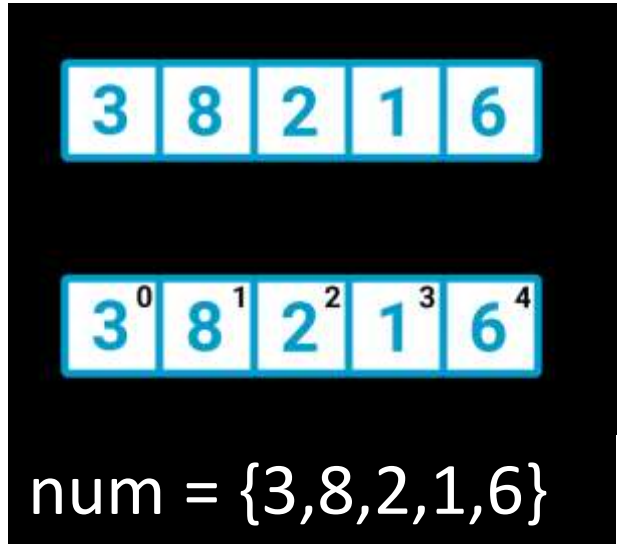
Un array es una colección de datos, de variables.

Tiene un nombre de variable único que representa a cada elemento dentro de él.

Es posible el acceso a cada elemento de un array a través de un número entero que se denomina índice.

La numeración de estos elementos dentro del array comienza en 0 (primer elemento del array) y finaliza en n-1 (último elemento del array) donde n es el tamaño completo de dicho array.

Los elementos dentro del array son guardados en posiciones de memoria de forma continua.

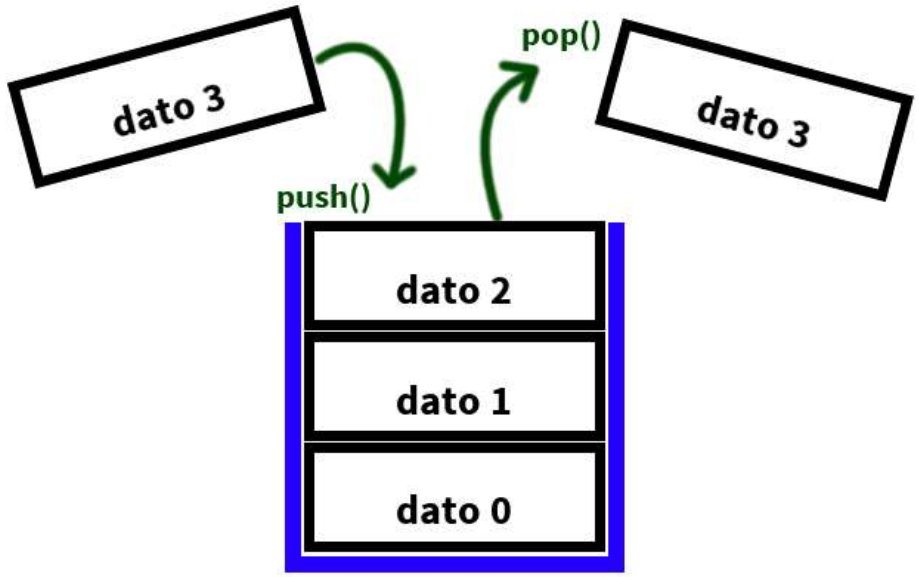


INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Estructura de datos – Pilas.

Una pila (stack) es una lista en la que el modo de acceso a sus elementos es de tipo LIFO (Last In First Out, último en entrar, primero en salir) que permite almacenar y recuperar datos.



En cada momento sólo se tiene acceso a la parte superior de la pila, es decir, al último objeto apilado.

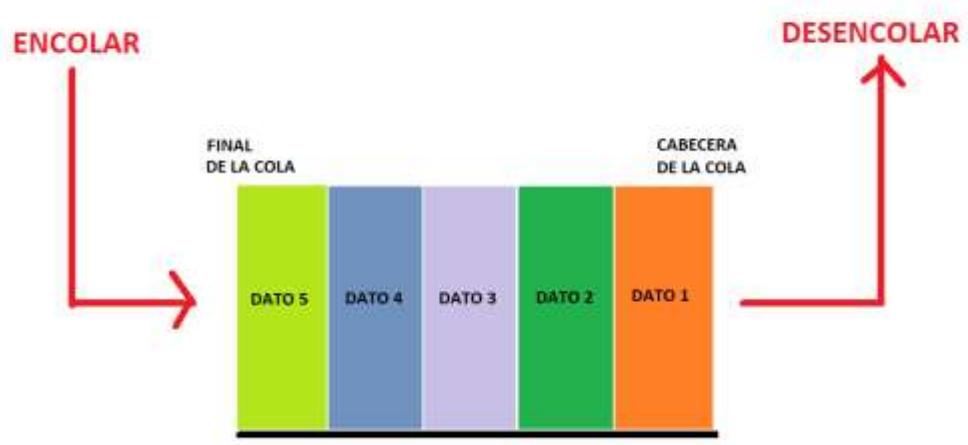
Para el manejo de los datos se cuenta con dos operaciones básicas: **apilar** (push), que coloca un objeto en la pila, y **desapilar** (pop), que retira el último elemento apilado.

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Estructura de datos – Colas.

Una cola es una estructura de datos, caracterizada por ser una secuencia de elementos en la que la operación de inserción push se realiza por un extremo y la operación de extracción pop por el otro. También se le llama estructura FIFO (First In First Out), debido a que el primer elemento en entrar será también el primero en salir.



INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Estructura de datos.

La elección de la estructura de datos adecuada es tan importante como diseñar un correcto flujo en nuestro código para resolver un problema.

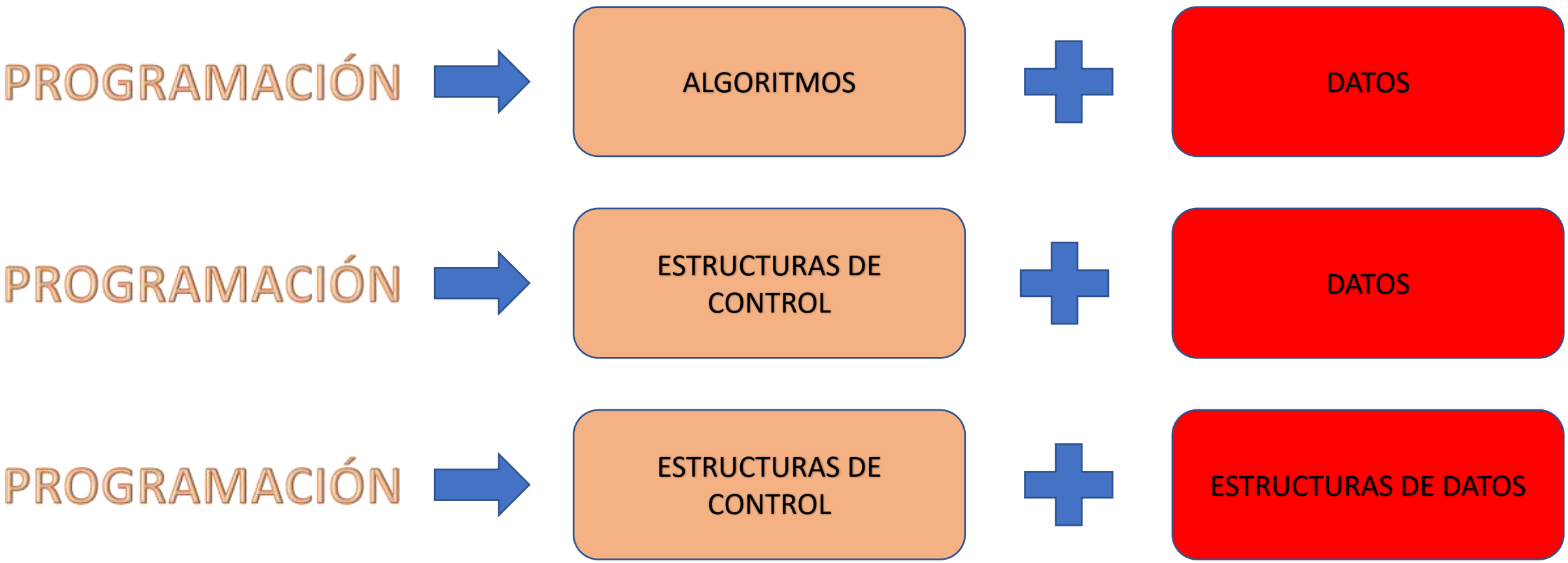
Python ofrece de forma “nativa” cuatro estructuras de datos:

- ❑ Listas
- ❑ Tuplas
- ❑ Conjuntos
- ❑ Diccionarios

Las **estructuras de datos** nos permiten el **agrupamiento de los datos** y las **estructuras de control** como while o for, **su procesamiento**.

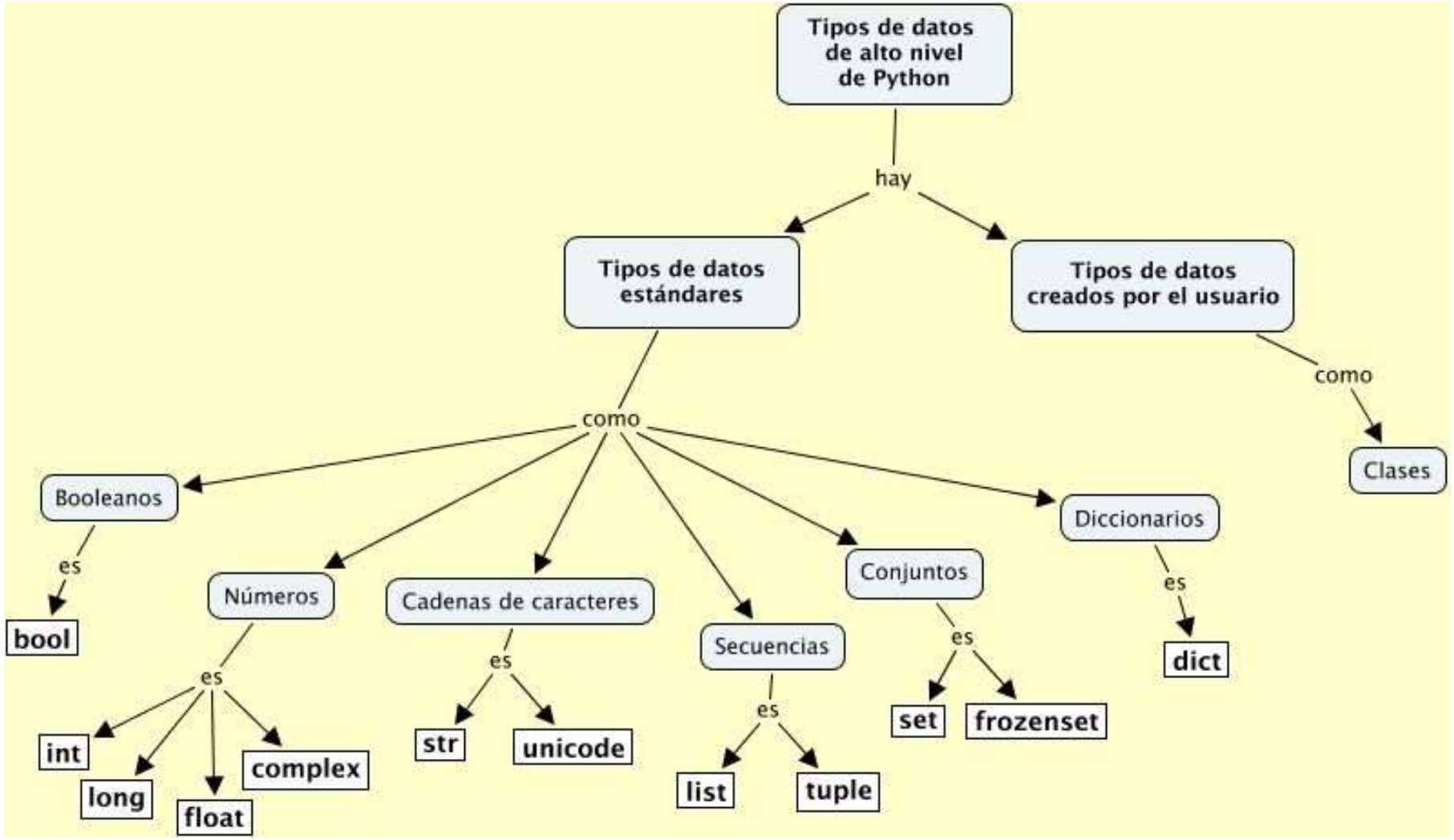
INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.



INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.



INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Listas

La **lista** es un tipo de colección ordenada. Sería equivalente a lo que en otros lenguajes se conoce por **arrays**, o **vectores**.

Las listas pueden contener **cualquier tipo de dato**: números, cadenas, booleanos, … y también listas.

Crear una lista es tan sencillo como indicar entre corchetes, y separados por comas, los valores que queremos incluir en la lista.

```
mi_lista = ['cadena de texto', 15, 2.8, 'otro dato', 25]
```

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Listas

También podemos crear una lista usando la función **list()**

```
mi_lista = list()      # Lista vacía
```

Podemos generar una lista con valores enteros con la función **range()**

```
mi_lista = list(range(7))      # Devuelve: [0,1,2,3,4,5,6]
```

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Listas

Se puede acceder a cada uno de los datos mediante su índice correspondiente, siendo 0 (cero), el índice del primer elemento.

```
mi_lista = ['cadena de texto', 15, 2.8, 'otro dato', 25]

print mi_lista[1] # Salida: 15
```

Si intentamos acceder a una posición más allá de los elementos de mi lista Python comunicará un error, pues ese índice de posición no existe.

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Listas

Si se utiliza un número negativo como índice, esto se traduce en que el índice empieza a contar desde el final, hacia la izquierda; es decir, con [-1] accederíamos al último elemento de la lista, con [-2] al penúltimo, con [-3], al antepenúltimo, y así sucesivamente.

```
mi_lista = ['cadena de texto', 15, 2.8, 'otro dato', 25]

print mi_lista[-2] # Salida: otro dato
```

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Listas

Si en lugar de un número escribimos dos números inicio y fin separados por dos puntos **[inicio:fin]** Python interpretará que queremos una lista que vaya desde la posición inicio a la posición fin, **sin incluir este último**. Este valor fin no es el índice del último elemento, sino el valor que no puede superar ninguna posición de las que estamos recuperando.

```
mi_lista = ['cadena de texto', 15, 2.8, 'otro dato', 25]

print mi_lista[1:4] # Devuelve: [15, 2.8, 'otro dato']
```

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Listas

Las lista permiten modificar los datos una vez creados:

```
mi_lista = ['cadena de texto', 15, 2.8, 'otro dato', 25]

mi_lista[2] = 3.8 # el tercer elemento ahora es 3.8
```

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Listas

Las listas permiten agregar nuevos valores al final de la lista con el método **lista.append(x)** :

```
mi_lista.append('Nuevo Dato')
```

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Listas

Eliminar un elemento de la lista mediante la instrucción:

del lista[índice]

```
mi_lista = ['cadena de texto', 15, 2.8, 'otro dato', 25]

del mi_lista [3] # elimina 'otro dato'
```

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Listas

Para saber el número de elementos de una lista usamos la instrucción:
len(lista)

```
mi_lista = ['cadena de texto', 15, 2.8, 'otro dato', 25]

elementos = len(mi_lista)
```

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Listas

Las listas ofrecen mecanismos más cómodos para ser modificadas a través de las funciones:

len(list) Devuelve el número de elementos de una lista.

list.append(x) Añade el elemento x al final de la lista.

list.extend(iterable) Añade los elementos del iterable a la lista.

list.insert(i, x) Inserta el elemento x en la posición i.

list.remove(x) Elimina de la lista el primer (y solo el primer) elemento con el valor de x.

list.pop([i]) Devuelve el valor en la posición i y lo elimina de la lista. Si no se especifica la posición, se utiliza el último elemento de la lista.

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Listas

list.clear() Vacía la lista eliminando todos sus elementos.

list.count(x) Devuelve el número de veces que se encontró el elemento x en la lista. Si el elemento no aparece en la lista el valor devuelto es cero.

list.index(x[, start[, stop]]) Devuelve la posición en la que se encontró la primera ocurrencia del elemento x. Si se especifican, start y stop definen las posiciones de inicio y fin de una sublista en la que buscar.

list.reverse() Invierte la lista. Esta función trabaja sobre la propia lista desde la que se invoca el método, no sobre una copia.

list.copy() Devuelve una copia de la lista.

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Tuplas

Una tupla es una variable que permite almacenar varios datos inmutables (no pueden ser modificados una vez creados) de tipos diferentes.

Para crear una tupla, basta asignar a una variable una secuencia de elementos separados por comas. Podemos enmarcarlos con paréntesis o no:

```
mi_tupla = ('cadena de texto', 15, 2.8, 'otro dato', 25)
```

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Tuplas

Es posible crear una tupla a partir de los otros tipos de secuencia: listas o cadenas. Para esto haremos uso de la función **tuple()**

```
mi_tupla = tuple(range(4))        # Devuelve: (0,1,2,3)

mi_cadena = “Hola”
mi_tupla = tuple(mi_cadena)    # Devuelve: (´H`, ´o`, ´l`, ´a`)
```

Las tuplas también pueden anidarse, incluyendo tuplas dentro de tuplas.

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Tuplas

Se puede acceder a cada uno de los datos mediante su índice correspondiente, siendo 0 (cero), el índice del primer elemento:

```
mi_tupla = ('cadena de texto', 15, 2.8, 'otro dato', 25)

print mi_tupla[1] # Salida: 15
```

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Tuplas

También se puede acceder a una porción de la tupla, indicando (opcionalmente) desde el índice de inicio hasta el índice de fin:

```
mi_tupla = ('cadena de texto', 15, 2.8, 'otro dato', 25)

print mi_tupla[1:4] # Devuelve: (15, 2.8, 'otro dato')
print mi_tupla[3:] # Devuelve: ('otro dato', 25)
print mi_tupla[:2] # Devuelve: ('cadena de texto', 15)
```

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Tuplas

Otra forma de acceder a la tupla de forma inversa (de atrás hacia adelante), es colocando un índice negativo:

```
mi_tupla = ('cadena de texto', 15, 2.8, 'otro dato', 25)

print mi_tupla[-1] # Salida: 25
print mi_tupla[-2] # Salida: otro dato
```

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Desempaquetado / Empaquetado de secuencias

Estamos habituados a ver código de asignación de valores a variables con una única variable sobre la cual recae el valor a asignar.

myVariable = ‘valor’

Es posible realizar una asignación múltiple a partir de estructuras de tipo secuencia (cadenas, listas y tuplas).

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Desempaquetado / Empaquetado de secuencias

Si se tiene una tupla de longitud k, se puede asignar la tupla a k variables distintas y en cada variable quedará una de las componentes de la tupla. **A esta operación se la denomina desempaquetado de secuencias.**

depredador, presa = (“aguila”, “liebre”)

Obtenemos: depredador = águila; presa = liebre

Si se tiene una tupla de longitud k, se puede asignar la tupla a k variables distintas y en cada variable quedará una de las componentes de la tupla. A esta operación se la denomina *desempaquetado*

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Desempaquetado / Empaquetado de secuencias

Si a una variable se le asigna una secuencia de valores separados por comas, el valor de esa variable será la tupla formada por todos los valores asignados. **A esta operación se la denomina empaquetado de secuencias**

mi_tupla = 34,5.6,86

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Diccionarios

Los diccionarios, también llamados matrices asociativas, deben su nombre a que son colecciones que relacionan una **clave** y un **valor**.
Los pares clave-valor están separados por dos puntos, las parejas por comas y todo el conjunto se encierra entre llaves.

mi_diccionario = {'clave_1': valor_1, 'clave_2': valor_2, 'clave_7': valor_7}

En otros lenguajes, a los diccionarios se los llama arrays asociativos, matrices asociativas, o también tablas de hash.

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Diccionarios

Por ejemplo, veamos un diccionario de películas y directores:

d = {“Love Actually “: “Richard Curtis”, “Kill Bill”: “Tarantino”, “Amélie”: “Jean-Pierre Jeunet”}

El primer valor se trata de la clave y el segundo del valor asociado a la clave.

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Diccionarios

También podemos crear un diccionario con la función **dict()**

```
mi_diccionario =dict({'clave_1': valor_1, 'clave_2': valor_2, 'clave_7': valor_7})
```

Las **claves son únicas** dentro de un diccionario, es decir que no puede haber un diccionario que tenga dos veces la misma clave, **si se asigna un valor a una clave ya existente, se reemplaza el valor anterior**.

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Diccionarios

O bien inicializar el diccionario vacío y luego agregar los valores de a uno o de a muchos.

```
materias = {}
materias["lunes"] = [6103, 7540]
materias["martes"] = [6201]
materias["miércoles"] = [6103, 7540]
materias["jueves"] = []
materias["viernes"] = [6201]
```

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Diccionarios

Para acceder al valor asociado a una determinada clave:

```
mi_diccionario = {'clave_1': valor_1, 'clave_2': valor_2}

print mi_diccionario['clave_2’]   # Salida: valor_2
```

Sin embargo, **esto falla si se provee una clave que no está en el diccionario**.

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Diccionarios

Para comprobar la existencia de una clave podemos usar la función **get()** o el operador **in**

```
mi_diccionario = {'clave_1': valor_1, 'clave_2': valor_2}

'clave_1' in mi_diccionario                # Salida: True

mi_diccionario.get('clave_1')              # Salida: True
```

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Diccionarios

Un diccionario permite eliminar cualquier entrada con la función:

del()

mi_diccionario = {'clave_1': valor_1, 'clave_2': valor_2, 'clave_7': valor_7}

del(mi_diccionario['clave_2'])

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Diccionarios

Al igual que las listas, el diccionario permite modificar los valores

```
mi_diccionario = {'clave_1': valor_1, 'clave_2': valor_2, 'clave_7': valor_7}

mi_diccionario['clave_1'] = 'Nuevo Valor'
```

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Diccionarios

Existen diversas formas de recorrer un diccionario.

Obtendremos un listado de todas las claves:

```
d = {'clave_1': valor_1, 'clave_2': valor_2, 'clave_3': valor_3, 'clave_4': valor_4 }

for k in d:
        print(k)
```

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Diccionarios

Si queremos mostrar los valores:

```
d = {'clave_1': valor_1, 'clave_2': valor_2, 'clave_3': valor_3, 'clave_4':
valor_4 }

for k in d:
        print(d[k])
```

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Diccionarios

Podemos mostrar los valores de manera más eficiente accediendo a los valores mediante **values()**:

```
d = {'clave_1': valor_1, 'clave_2': valor_2, 'clave_3': valor_3, 'clave_4': valor_4 }

for v in d.values():
        print(v)
```

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Diccionarios

Si deseamos mostrar tanto clave como valor:

```
d = {'clave_1': valor_1, 'clave_2': valor_2, 'clave_3': valor_3, 'clave_4': valor_4 }

for k in d:
        print(k, '→' , d[k])
```

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Diccionarios

Disponemos de **items()** para mostrar tanto clave como valor de manera más eficiente :

```
d = {'clave_1': valor_1, 'clave_2': valor_2, 'clave_3': valor_3, 'clave_4': valor_4 }

for k, v in d.ítems():
        print(k, '→’ , d[k])
```

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Diccionarios

La diferencia principal entre los diccionarios y las listas o las tuplas es que a los valores almacenados en un diccionario **se les accede** no por su índice, porque de hecho no tienen orden, sino **por su clave**, utilizando de nuevo el operador [].

En general, los diccionarios sirven para crear bases de datos muy simples, en las que la clave es el identificador del elemento, y el valor son todos los datos del elemento a considerar.

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Conjuntos

Un conjunto, es una colección no ordenada, ni indexada y sin elementos repetidos.

Para crear un conjunto especificamos sus elementos entre llaves:

```
s = {1, 2, 3, 4}
```

Al igual que otras colecciones, sus miembros pueden ser de diversos tipos:

```
s = {True, 3.14, False, "Hola mundo", (1, 2)}
```

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Conjuntos

De la misma forma podemos obtener un conjunto a partir de una secuencia, usando la función **set()**

```
s = set()
s2 = set(range(10))
```

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Conjuntos

Los conjuntos son objetos **mutables**. Con **add()** y **discard()** podemos añadir y remover un elemento indicándolo como argumento.

```
s = {1, 2, 3, 4}
s.add(5)
s.discard(2)

# Obtenemos: {1, 3, 4, 5}
```

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Conjuntos

La función **clear()** elimina todos los elementos.

```
s = {1, 2, 3, 4}
s.clear()
```

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Conjuntos

Sus operaciones principales son: **unión**, **intersección** y **diferencia**.

La **unión** se realiza con el carácter **|** y retorna un conjunto que contiene los elementos que se encuentran en al menos uno de los dos conjuntos involucrados en la operación.

```
a = {1, 2, 3, 4}
b = {3, 4, 5, 6}
a | b
# Devuelve: {1, 2, 3, 4, 5, 6}
```

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Conjuntos

La **intersección** se realiza con el operador **&**, y retorna un nuevo conjunto con los elementos que se encuentran en ambos.

```
a = {1, 2, 3, 4}
b = {3, 4, 5, 6}
a & b
# Devuelve: {3, 4}
```

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

Conjuntos

La **diferencia**, por último, retorna un nuevo conjunto que contiene los elementos de a que no están en b.

```
a = {1, 2, 3, 4}
b = {2, 3}
a - b
# Devuelve: {1, 4}
```

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

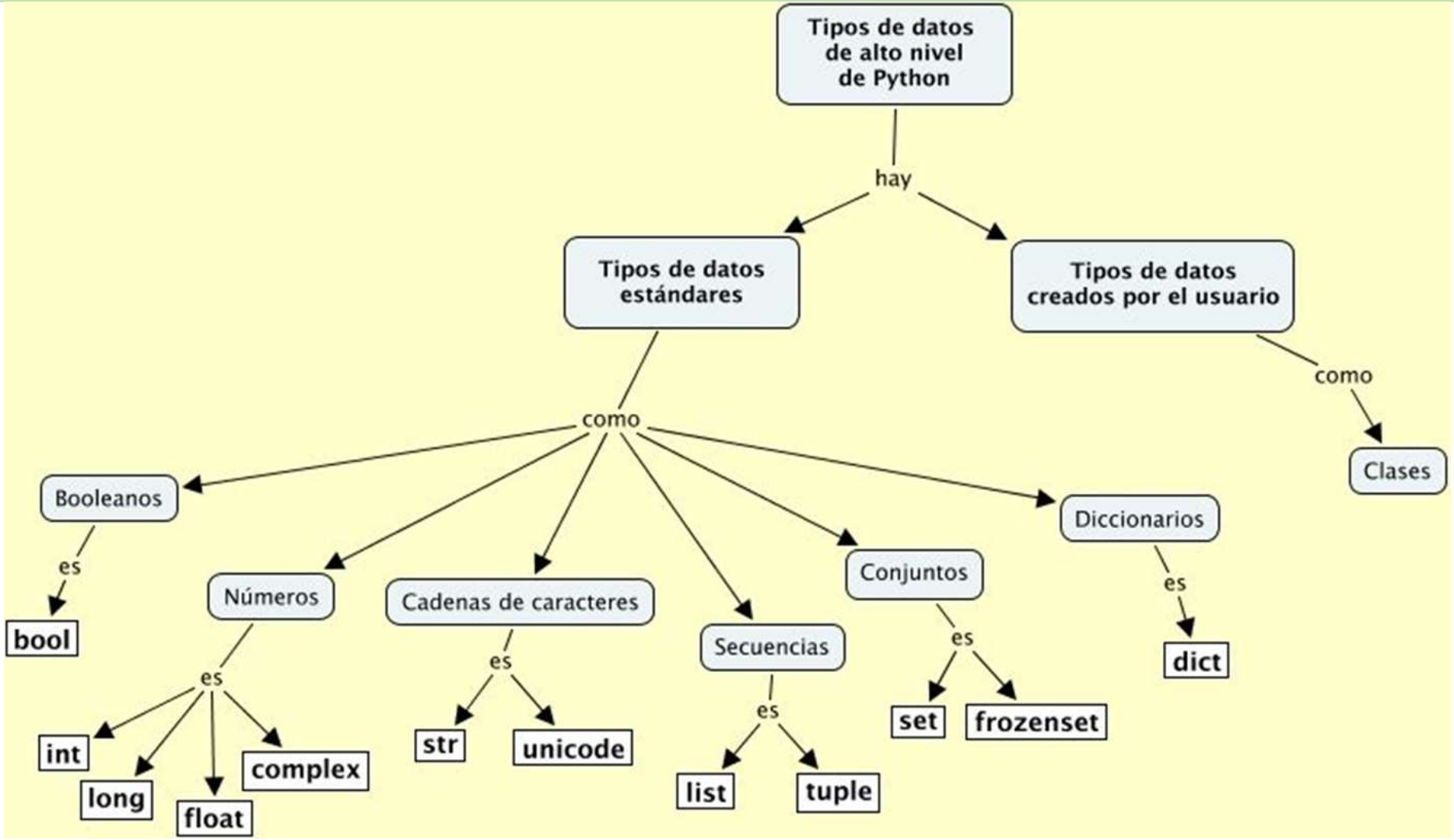
Conjuntos inmutables

frozenset es una implementación similar a **set** pero **inmutable**. Es decir, comparte todas las operaciones de conjuntos a excepción de aquellas que implican alterar sus elementos (add(), discard(), etc.). La diferencia es análoga a la existente entre una lista y una tupla.

```
a = frozenset({1, 2, 3})
b = frozenset({3, 4, 5})
```

INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.



INTRODUCCIÓN A LA PROGRAMACIÓN

Estructuras de Datos.

“Fin del tema”