python™

Estructuras de Datos

CTIC - UNI

Universidad Nacional de Ingeniería

Estructuras de Datos

- Las Estructuras de Datos son formas particulares de almacenar datos para hacer alguna operación más fácil o más eficiente. Es decir, están sintonizados para ciertas tareas.
- Las estructuras de datos son adecuadas para resolver ciertos problemas, y son frecuentemente asociadas con algoritmos.

Tipos de Estructuras de Datos

Aproximadamente dos tipos de estructuras de datos:

- *Estructuras de datos incorporadas*, estructuras de datos que son tan comunes que Python los proporciona de forma predeterminada.
- *Estructuras de datos definidas por el usuario* (clases en programación orientada a objetos) que están diseñadas para una tarea particular.

Estructuras de datos incorporadas Python

- Python viene con un conjunto general de estructuras de datos integradas:
  - strings
  - listas
  - tuplas
  - conjuntos
  - diccionarios
  - otros

python™

Listas

CTIC - UNI

Universidad Nacional de Ingeniería

La estructura de datos lista Python

- Una lista es una secuencia ordenada de items.
- Una cadena es solo un tipo particular de lista.

Creando una Lista

- Como todas las estructuras de datos, las listas tienen un **constructor**, llamado igual que la estructura de datos. Toma una estructura de datos iterable y **agrega cada elemento** a la lista.
- También tiene una forma directa, el uso de corchetes [ ] para indicar elementos explícitamente.

Lista Vacía

- Una lista es una colección ordenada de objetos, similar al array dinámico empleado en otros lenguajes de programación.

- Puede contener diferentes tipos de objetos, es mutable y Python nos ofrece una serie de funciones y métodos integrados para realizar diferentes tipos de operaciones.

- Inicialización de una lista:

```
lista = [ ]
```

Creando una Lista

- lista=[1, 2, 'a', 3.14159]
- lista_dds=['LUN','MAR','MIE','JUE','VIE']
- lista_de_coleccion=list('Hola')
- lista_de_listas=[[1,2,3] , ['a','b','c']]

Similaridades con strings

- concatenacion/+ (pero solo de listas)
- repetición /*
- indexación (el operador [ ])
- slicing ([:])
- menbresía (el operador in)
- len (el operador length)

Operadores

```
[1, 2, 3] + [4] ⇒ [1, 2, 3, 4]

[1, 2, 3] * 2 ⇒ [1, 2, 3, 1, 2, 3]

1 in [1, 2, 3] ⇒ True

[1, 2, 3] < [1, 2, 4] ⇒ True
```

compara índice a índice, la primera diferencia determina el resultado.

diferencias entre listas y strings

- Las listas pueden contener una mezcla de cualquier objeto python, las cadenas solo pueden contener caracteres
  - 1,"bill",1.2345, True
- Las listas son mutables, sus valores pueden ser cambiados, mientras los strings son inmutables.
- Las listas son diseñadas con [ ], con elementos separados por comas. En el caso de strings use " " or ' '

Indexación

- puede ser un poco confuso, ¿qué significa [ ], una lista o un índice?

```
[1, 2, 3][1] ⇒ 2
```

- El contexto resuelve el problema. El índice siempre aparece al final de una expresión y está precedido por algo (una variable, una secuencia)

Lista de Listas

```
mi_lista = ['a', [1, 2, 3], 'z']
```

- ¿Cuál es el segundo elemento (índice 1) de esa lista? Otra lista.

```
mi_lista[1][0] # de izq. a derecha
 mi_lista[1] ⇒ [1, 2, 3]
 [1, 2, 3][0] ⇒ 1
```

Funciones de Listas

- **min(lista)**
  - calcula el mínimo en la lista.
- **max(lista)**
  - calcula el mínimo en la lista.
- **sum(lista)**
  - calcula la suma de una lista. Solo valores numéricos.
- **len(lista)**
  - Calcula la longitud de una lista

Iteracion

Puede recorrer los elementos de una lista como se hacen con una cadena:

```
>>> mi_lista=[1,3,4,8]
>>> for item in mi_lista:
        print(item, end='');


1348
>>>
```

Range

- Hemos visto la función de rango antes. Genera una secuencia de enteros.
- Lo que genera es una lista con esa secuencia:
- `Mi_Lista = range(1,5)`

  `Mi_Lista es [1,2,3,4]`

  `Lo comprobamos consultando`

  `2 in Mi_Lista`

  `4 in Mi_Lista`

python™

Mutable

CTIC - UNI

Universidad Nacional de Ingeniería

Cambiar los contenidos de un objeto

- strings son inmutables. Una vez creado, el contenido del objeto no puede ser cambiado. Se pueden crear nuevos objetos para reflejar un cambio, pero el objeto en sí no se puede cambiar

```
mi_str = 'abc'
mi_str[0] = 'z'   # incorrecto!
# en su lugar, crea una nueva str
nuevo_str = mi_str.replace('a','z')
```

Las listas son mutables

A diferencia de las cadenas, las listas son **mutables**. ¡Puedes **cambiar** el contenido del objeto!

```
mi_lista = [1, 2, 3]
mi_lista[0] = 127
print(mi_lista) ⇒ [127, 2, 3]
```

Métodos para el manejo de Listas

- Recuerde, una función es un programa pequeño (como len) que toma algunos argumentos, los elementos entre paréntesis y devuelve algún valor.
- Un método es una función llamada de una manera especial, la llamada **punto**. Se llama en el contexto de un objeto (o una variable asociada con un objeto)

Las listas tienen métodos

```
mi_lista = ['a',1,True]
mi_lista.append('z')
```

argumentos para el método

El objeto con el que estamos llamando al método.

el nombre del método

Algunos métodos nuevos

- Una lista es mutable y puede cambiar:
  - `mi_lista[0]='a'#asignación de índice`
  - `mi_lista.append(), mi_lista.extend()`
  - `mi_lista.pop()`
  - `mi_lista.insert(), mi_lista.remove()`
  - `mi_lista.sort()`
  - `mi_lista.reverse()`

Más acerca de la lista de métodos

- La mayoría de estos métodos no devuelven un valor.
- Esto se debe a que las listas son mutables, por lo que los métodos modifican la lista directamente. No hay necesidad de devolver nada.
- Puede ser confuso.

Listas

- **list.append(x)**

  Añade un elemento al final de la lista

- **list.count(x)**

  Devuelve el número de veces que x aparece en la lista.

- **list.extend(L)**

  Amplía la lista agregando todos los elementos de la lista dada

Listas

- **list.index(x)**
  Devuelve el índice en la lista del primer elemento cuyo valor es x.

- **list.insert(i, x)**
  Inserta un elemento en una posición dada. El primer argumento es el índice del elemento antes de insertar

-

Listas

- **list.pop([i])**

  Elimina el elemento en la posición dada en la lista y lo devuelve. Si no se especifica ningún índice, pop () elimina y devuelve el último elemento de la lista.

- **list.remove(x)**

  Elimina el primer elemento de la lista cuyo valor es x. Es un error si no hay tal artículo.

- **list.reverse()**

  Invierte los elementos de la lista.

Listas

- **random.shuffle(lista)**
- Se utiliza para barajar la secuencia en su lugar, es decir, cambia la posición de los elementos en una lista.

- **list.sort(x)**

  Ordena los elementos de la lista.

Split

- El método de división de cadena **Split** genera una secuencia de caracteres al dividir la cadena en ciertos caracteres de división.
- ***Devuelve una lista***

```
split_lista = 'esta es una prueba'.split()
split_lista
      ⇒ ['esta', 'es', 'una', 'prueba']
```

Resultados Inesperados

```
mi_lista = [4, 7, 1, 2]
mi_lista = mi_lista.sort()
mi_lista ⇒ None    # que pasó?
```

Lo que sucedió fue que la operación de clasificación cambió el orden de la lista en su lugar (lado derecho de la asignación). Luego el método de clasificación devolvió **None**, que fue asignado a la variable. La lista se perdió y **None** es ahora el valor de la variable.

Segmentos de Listas

`milista[indiceinicial:indice_final]` evalúa a una sublista de la lista original.

- **`milista[indice]`** evalúa a un elemento de la lista original

- Los argumentos son como los de la función range
  - **`milista[inicio:fin:paso]`**
  - el índice inicial es inclusivo, el índice final es exclusivo
  - *Los 3 índices son opcionales.*
- Se puede asignar a una porción:

```
milista[s:e] = tulista
```

Ejemplos de Segmentación en Listas

```
test_lista = ['e0', 'e1', 'e2', 'e3', 'e4', 'e5', 'e6']
```

Desde e2 hasta el final de la lista:

```
test_lista[2:]
```

Desde el principio hasta (pero sin incluir) e5:

```
test_lista[:5]
```

Último elemento:

```
test_lista[-1]
```

Últimos cuatro elementos:

```
test_lista[-4:]
```

Todo excepto los últimos tres elementos:

```
test_lista[:-3]
```

Invierte la lista:

```
test_lista[::-1]
```

Obtener una copia de toda la lista:

```
test_lista[:]
```