Lenguaje de Programación Python

Listas

Dr. Mario Marcelo Berón
Argentina Programa
Universidad Nacional de San Luis

Listas: Tipos de Datos Secuencia

Los tipos de dato secuencia en Python incluyen:

- Listas
- Strings
- Tuplas
- Objetos retornados por *range*

Operaciones comunes:

- Indexación
- Slicing
- Loops
- len
- in
- not in

Listas: Tipos Mutables e Inmutables

- Tipo Mutable: se pueden agregar, eliminar o cambiar valores.
- Tipo Inmutable: no es mutable.

Listas: Características

- Contienen valores lo que hace que los programas administren grandes cantidades de datos de forma sencilla.
- Contiene valores en una secuencia ordenada.
- Las listas comienzan y finalizan con corchetes.
- Los valores dentro de la lista se denominan ítems.
- Los ítems se separan con comas.
- Las listas son heterogéneas (también pueden contener listas).

Ejemplos

[1,2,3,4]
["Juan",1,Çarlos",[1,2,3]]

Listas: Características

- Los ítems de la lista se acceden por posición. Ejemplo: l[0] hace referencia al primer elemento de la lista l, l[1] al ítem que se encuentra en la posición 1 y así siguiendo.
- Los índices deben ser valores enteros no pueden ser flotantes.
- Se pueden usar índices negativos. El índice -1 se refiere al último elemento de la lista, el -2 al anteúltimo y así siguiendo.
- Se pueden obtener porciones de una lista utilizando los *slice*. En un slice el primer elemento indica dónde comienza y el último donde termina (no incluye el último). Ej: l[1 : 3] devuelve una lista que contiene los ítems l[1],l[2].

Listas: Características

- En un slice se pueden obviar los índices:
  - Si se obvia el ínidice inicial se toma como valor de éste 0.
  - Si se obvia el índice final se toma como valor de este la longitud de la lista.
  - Si se obvian los dos el resultado es la lista en sí misma.

Listas: Características

Ejemplo

```
>>> l = [10,20,30,40,50]
>>> l[:3]
 >>> [10,20]
>>> l[2:]
 >>> [20,30,40,50]
>>> l[:]
 >>> [10,20,30,40,50]
```

Listas: Funciones

- *len* retorna como resultado la longitud de la lista.
- Para cambiar un elemento de la lista se utiliza la operación de asignación. Ejemplo: l[0]=20.
- El operador + permite concatener listas.
- El operador * permite replicar una lista.
- *del* elimina valores de la lista en el índice indicado. Ejemplo: del l[0] elimina el ítem que se encuentra en la posición 0 de la lista l.

Listas: Recorrido

Recorrido con range:

```
for i in range(4):
  print(i)
```

Recorrido con una lista explícita:

```
for i in [0,1,2,3]:
  print(i)
```

Estrategia usada comunmente para recorrer listas:

```
for i in range(len(lista)):
  print(lista[i])
```

Listas: in y not in

- *in*: permite determinar si un elemento se encuentra en la lista.
- *not in*: permite determinar si un elemento no se encuentra en la lista.

Ejemplo

```
l=[1,2,"hola",[10,20],"otro"]

"hola" in l
True

[10,20] in l
True

"pepe" not in l
True
```

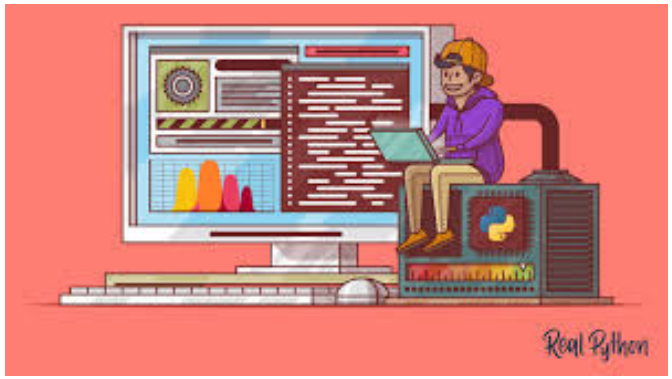
Listas: Desempaquetado de Ítems

Se puede realizar una asignación a múltiples variables con los valores de una lista.

Ejemplo

```
l=["Carlos","Alberto","Domingo"]
c,a,d=l
c
Carlos
a
Alberto
d
Domingo
```

Listas: enumerate



La función *enumerate* retorna dos valores: el índice de un ítem y el ítem en sí mismo.

Ejemplo

```
p=["Carlos","Antonio","Diego","Gabriel"]
for ind,it in enumerate(p):
  print("Indice:",ind,"␣Item:",it)
```

Listas: random.choice(), random.shuffle()

- *random.choice*: retorna un ítem de la lista seleccionado aleatoriamente.
- *random.shuffle*: reordena los ítems de la lista.

Ejemplo

```
import random
l=[10,20,30,40,50]
v=random.choice(l)
30
random.shuffle(l)
l
[30,40,10,20,50]
```

Listas: Asignación Aumentada

- El operador += puede hacer la concatenación de strings y listas.
- El operador *= puede replicar strigns y listas.

Ejemplo

```
l =[10,20,30,40]
l +=[100,200]
l
[10, 20, 30, 40, 100, 200]
l *2
[10, 20, 30, 40, 100, 200, 10, 20, 30, 40, 100, 200]
```

Listas: Métodos

- *index*: se le pasa un valor como parámetro si ese valor se encuentra en la lista (el objeto receptor) retorna como resultado su índice. Caso contrario produce una excepción *ValueError*.
- *append*: incorpora ítems al final de la lista.
- *insert*: incorpora ítems en una posición determinada por el usuario. Este método recibe dos parámetros, el primero es la posición donde se desea insertar el ítem y el segundo es el ítem en sí mismo.

Listas: Métodos Útiles

index

```
>>> l=[10,"Pedro", 20,"Maria"]
>>> l.index("Pedro")
1
```

append

```
>>> l
[10, 'Pedro', 20, 'Maria']
>>> l.append("Institución")
>>> l
[10, 'Pedro', 20, 'Maria', 'Institución']
```

Listas: Métodos Útiles

insert

```
>>> l
[10, 'Pedro', 20, 'Maria', 'Institución']
>>> l
[10,'Pedro',20,'Maria','Institución']
>>> l.insert(2,"Educativa")
>>> l
[10,'Pedro','Educativa',20,'Maria','Institución']
```

Listas: Métodos

- *remove*: elimina el valor pasado como parámetro de la lista. Eliminar un ítem que no está en la lista produce una excepción *ValueError*.
- *sort*: ordena los ítems de una lista. Si este método recibe como parámetro True ordena la lista en orden inverso.
  - La lista se ordena en el lugar.
  - No se puede clasificar una lista que tenga números y strings porque no se puede realizar la comparación.
  - El orden que utiliza este método hace uso de: ASCIIbetical order con lo cual las mayúsculas van antes que las minúsculas.
- *reverse*: inverte una lista.

Listas: Métodos Útiles

remove

```
>>> l
[10, 'Pedro', 'Educativa',20,'Maria', 'Institución']
>>> l.remove(20)
>>> l
[10, 'Pedro', 'Educativa', 'Maria', 'Institución']
```

Listas: Métodos Útiles

sort

```
>>> l=[-2,10,-100,20]
>>> l.sort()
>>> l
[-100, -2, 10, 20]
```

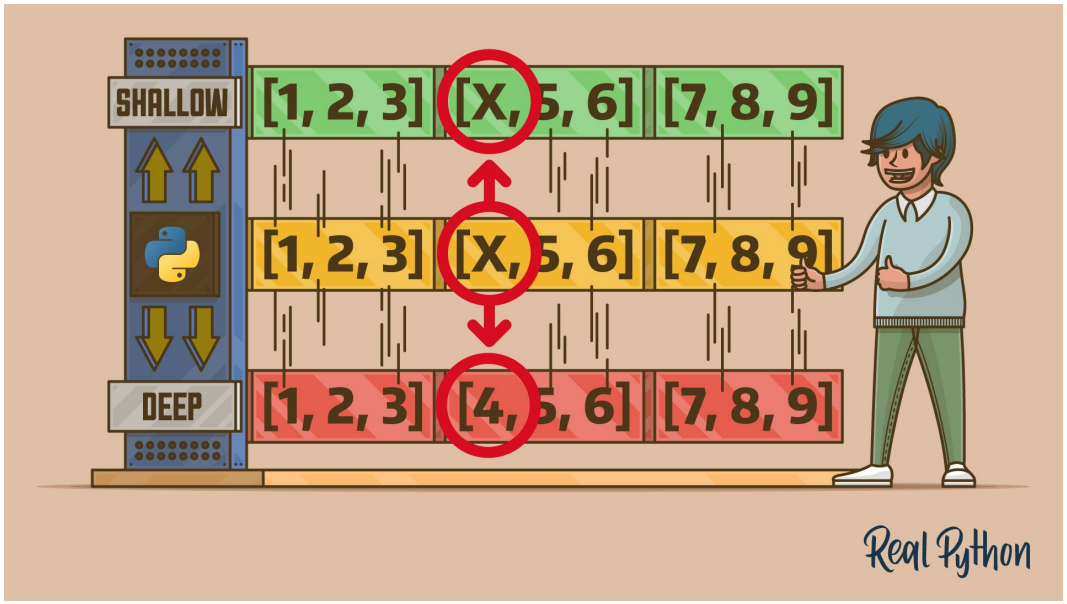
reverse

```
>>> l # [-100, -2, 10, 20]
>>> l.reverse()
>>> l
[20, 10, -2, -100]
```

Listas: id

Una forma de detectar alias es a través del uso de la función *id*. Esta función retorna como resultado la dirección de memoria del objeto que recibe como parámetro. Cuando el objeto es mutable y se produce una asignación es decir una copia de dirección entonces la dirección de memoria de las dos variables coincidirán.

Listas: copy y deepcopy



En muchas situaciones es necesario hacer copias "diferentes"de objetos mutables para alcanzar este objetivo, Python provee dos funciones que facilitan la tarea en el módulo copy:

- *copy*: crea una copia de una lista (o diccionario).
- *deepcopy*: crea una copia de una lista (o diccionario) que contiene ítems mutables (ejemplo una lista que contiene listas).

Listas: Métodos Útiles

copy

```
>>> nl=l.copy()
>>> nl
[20, 10, -2, -100]
```

deepcopy

```
>>> l #[20, 10, -2, -100]
>>> ol=copy.deepcopy(l)
>>> ol
[20, 10, -2, -100]
```

Conclusión

- La lista es un tipo de dato útil que permite escribir código que permite modificar un número variable de valores almacenados en una variable.
- Las listas es un tipo de dato secuencia que es mutable, es decir que su contenido se puede cambiar.
- Las variables almacenan referencias.
- *copy* y *deepcopy* se utilizan para realizar una copia del objetos mutables sin modificar el original.