Decisiones Generalizadas

Any, All y Aplicaciones

Introducción

- Hemos visto una parte del paradigma de programación funcional en Python, el cual puede ser fácilmente mezclado con paradigmas imperativos, aprovechando la disminución de los cambios de estados de las variables.
- Así como se pueden generalizar los cómputos para contenedores, también es posible la generalización de decisiones.
- Any y All nos permiten coleccionar sentencias declarativas o valores de verdad para tomar una decisión.
- En estas diapositivas se presentan ejemplos que encadenan conceptos vistos de funcional con las funciones Any y All.

All y Any

- Las funciones **all** y **any** se aplican sobre un iterable (una estructura que podamos recorrer de forma iterativa, como una lista o un conjunto) y devuelven *True* si todos los elementos son *True* (en el caso de la función **all**) o si algún elemento es True (en el caso de la función **any**), devolviéndose *False* en cualquier otra circunstancia.

```
print( all([True,False,True]) )
```

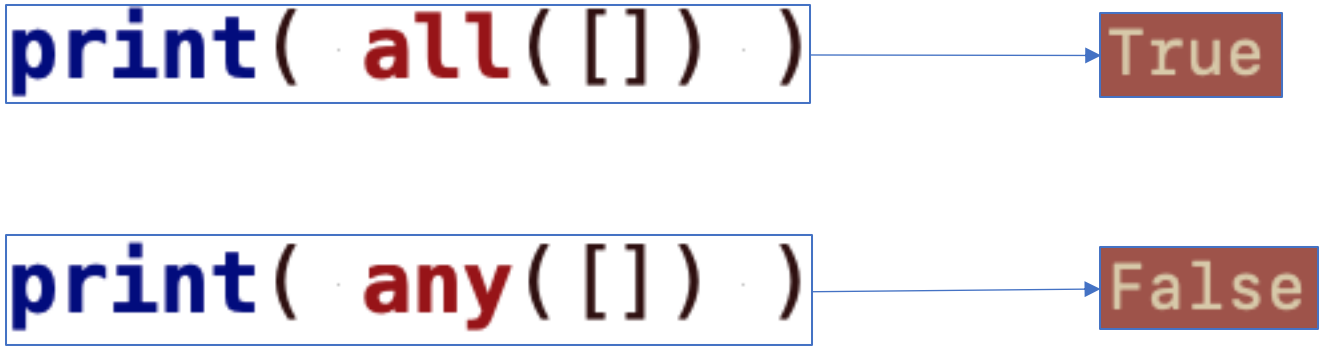
→ False

```
print( any([True,False,True]) )
```

→ True

All y Any

- Un caso especial se produce cuando el iterable en cuestión está vacío (cuando no tiene ningún elemento). En esta caso la función **all** devuelve *True* y la función **any** devuelve *False*:



Ejemplo All, Any y Programación Funcional

- Se recibe una lista de números enteros separados por espacios: Si todos los enteros son positivos, se debe revisar si algún entero es un número palíndromo como se muestra a continuación (https://en.wikipedia.org/wiki/Palindromic_number). Se imprime la palabra 'True' si se cumplen las condiciones o 'False' de lo contrario.

Ejemplo All, Any y Programación Funcional

- En este ejemplo se utilizan condicionales en una sola línea, conversión de contenedores, map, zip any y all.

```
1  info = [ int(input()), input().split(' ')  ]
2  print('True' if all( list(map(lambda x:x>0, list(map(int, info[1
   ])) )) ) and any(list(map( lambda x: x[0]==x[1] or x[0] == '5',
   list(zip(info[1], list(map( lambda x:x[-1:(len(x)+1)*-1:-1], info
   [1] )) ))))     else 'False'    )
```

Ejemplo All, Any y Programación Funcional

- Revisar con detenimiento la solución de **2 líneas**. La segunda línea está expresada como un párrafo.

```
1  info = [ int(input()), input().split(' ')  ]
2  print('True' if all( list(map(lambda x:x>0, list(map(int, info[1
   ])) )) ) and any(list(map( lambda x: x[0]==x[1] or x[0] == '5',
   list(zip(info[1], list(map( lambda x:x[-1:(len(x)+1)*-1:-1], info
   [1] )) ))))     else 'False'    )
```

Ejemplo 2 All, Any y Programación Funcional

La empresa ABCD tiene hasta 100 empleados. La compañía decide crear un número de identificación único UID para cada uno de sus empleados. La compañía le ha asignado la tarea de validar los UIDs generados aleatoriamente. Un UID válido debe cumplir con las siguientes reglas:

- Debe contener por lo menos dos letras mayúsculas en el alfabeto inglés.
- Debe tener por lo menos 3 dígitos.
- Contener únicamente dígitos alfanuméricos.
- No tener repeticiones.
- Contener exactamente 10 caracteres.

Ejemplo 2 All, Any y Programación Funcional

- Entradas y salidas esperadas:

```
uids = ['B1CD102354', 'B1CDEF2354']
```

```
Iválido
Válido
```

Ejemplo 2 All, Any y Programación Funcional

```
1  uids = ['B1CD102354', 'B1CDEF2354']
2  for uid in uids:
3      cond = []
4      #Por lo menos dos letras mayúsculas en el alfabeto inglés
5      cond.append( len(list( filter(lambda x: x.isupper(), list(uid)))) >= 2 )
6      #Por lo menos tres dígitos
7      cond.append( len(list( filter(lambda x: x.isdigit(), list(uid)))) >= 3 )
8      #Solamente dígitos alfanuméricos
9      cond.append( len(list( filter(lambda x: not(x.isalnum()), list(uid)))) == 0 )
10     #Que no existan repeticiones
11     cond.append( len(uid) == len(set(uid)) )
12     #Exactamente 10 caracteres
13     cond.append( len(uid) == 10 )
14     print('Válido') if all(cond) else print('Iválido')
```