

Taller de Programación

Clase 05: Listas

Daniela Opitz, Diego Caro
dopitz@udd.cl



Basada en presentaciones oficiales de libro Introduction to Programming in Python (Sedgewick, Wayne, Dondero).

Disponible en <https://introcs.cs.princeton.edu/python>

Outline

- Listas
- Procesamiento de datos con listas
- Break
- Continue


Listas

- **Lista:** secuencia de elementos de cualquier tipo.
- **Propósito:** facilitar el almacenamiento y procesamiento de datos.

Ejemplos:

- 52 cartas en un mazo
- 27 alumnos en una clase
- 8 millones de píxeles en una imagen
- 4 mil millones de nucleótidos en una base de ADN
- 86 mil millones de neuronas en el cerebro
- $6.02 \cdot 10^{23}$ partículas en un mol

<i>index</i>	<i>value</i>
0	2♥
1	6♠
2	A♦
3	A♥
...	
49	3♣
50	K♣
51	4♠



Elementos de una Lista

Corchetes
crean una lista

Elementos separados
por coma

Tercer elemento
en la lista

```
cartas = ['Diamante', 'Corazón', 'Pica', 'Trébol']  
print(cartas[2])
```

Acceso al i-ésimo
elemento

```
$ python3.7 lista.py  
Pica
```

Importante: El primer elemento está en la posición 0

Utilidad de una Lista

Sin una lista

Tedioso y propenso a generar errores

```
a0 = 0
a1 = 0
a2 = 0
a3 = 0
a4 = 3.1
a5 = 0
a6 = 0
a7 = 0
a8 = 5.2
a9 = 0

x = a4 + a8
```

Usando una lista

Sencillo


```
a = 10*[0]
a[4] = 3.1
a[8] = 5.2

x = a[4] + a[8]
```

**No es una multiplicación.
Repite 10 veces la lista [0]**

Utilidad de una Lista

```
a = 1000000*[0]  
a[234567] = 3.1  
a[891234] = 5.2  
  
x = a[234567] + a[891234]
```



Se puede escalar a millones de elementos!

Procesando Elementos en Listas

Utilizando ciclo for

```
1 meses = ['Enero', 'Febrero', 'Marzo', 'Abril', 'Mayo', 'Junio'  
2         , 'Julio', 'Agosto', 'Septiembre', 'Octubre', 'Noviembre'  
3         , 'Diciembre']  
4  
5 for mes in meses:  
6     print(mes)
```

Simple, menos propenso
a creación de bugs

Utilizando ciclo for + generación de posiciones

```
1 meses = ['Enero', 'Febrero', 'Marzo', 'Abril', 'Mayo', 'Junio'  
2         , 'Julio', 'Agosto', 'Septiembre', 'Octubre', 'Noviembre'  
3         , 'Diciembre']  
4  
5 n = len(meses) # tamaño lista meses  
6  
7 for i in range(n):  
8     print(meses[i])
```

1. Genera las posiciones en el arreglo
2. Recupera el elemento en la posición i

Procesando Elementos en Listas

- Los elementos de la lista se pueden acceder con el operador corchete []
- Si la posición del elemento es negativo, se accede desde el final.
- Si accedes una posición que no existe: **ERROR!**

0	1	2	3	4	5
2	3	5	7	11	
-5	-4	-3	-2	-1	

Input

```
1 L = [2, 3, 5, 7, 11]
2 print('L[0]', L[0])
3 print('L[1]', L[1])
4
5 print('L[-1]', L[-1])
6 print('L[-2]', L[-2])
7 print('L[99]', L[99])
```

Output

```
L[0] 2
L[1] 3
L[-1] 11
L[-2] 7
Traceback (most recent call last):
  File "lista-neg.py", line 7, in <module>
    print('L[99]', L[99])
IndexError: list index out of range
```

Error, programa se caerá. Lista L tiene 5 elementos.

Procesando Elementos en Listas

- Append: agregar nuevo elemento a la lista
- Concatenar: unir dos listas
- Obtener sublista: L[inicio:fin]
- Contiene: elem in L (devuelve True o False)

```
>>> 'a' in ['b','c','d','a']  
True
```

```
1 L = [11, 3, 5, 7, 2]  
2 print('L', L)  
3  
4 if 5 in L:  
5     print('cinco está en L')  
6  
7 # Actualizar elemento  
8 L[4] = 9999  
9 print('L[4]=9999', L)  
10  
11 # Agregar elemento a listas  
12 L.append(100) #Modifica lista  
13 print('L.append(100)', L)  
14  
15 # Concatenar lista  
16 L2 = L + [19, 17, 13] #Crea lista nueva  
17 print('L+[19, 17, 13]', L2)  
18  
19 # Sublista  
20 L3 = L[2:5] # Elementos 2,3 y 4  
21 print('L[2:5]', L3)
```

```
$ python3 ops.py  
L [11, 3, 5, 7, 2]  
cinco está en L  
L[4]=9999 [11, 3, 5, 7, 9999]  
L.append(100) [11, 3, 5, 7, 9999, 100]  
L+[19, 17, 13] [11, 3, 5, 7, 9999, 100, 19, 17, 13]  
L[2:5] [5, 7, 9999]
```

Procesando Elementos en Listas

P: ¿Cuándo usar for, o for + range?

R: La opción más simple es la adecuada ;)

```
1 # Calcula producto punto
2 x = [0.30, 0.60, 0.10]
3 y = [0.50, 0.10, 0.40]
4 total = 0.0
5 for i in range(len(x)):
6     total += x[i]*y[i]
7 print(total)
```

Variable Alias

- Si ambos elementos son listas, el operador de asignación crea un nuevo nombre a la variable

```
1 L = [1, 2, 3]
```

```
2 C = L
```

```
3 L[0] = 99
```

```
4
```

```
5 print(L)
```

```
6 print(C)
```



Crea un alias de la lista

Importante: El operador de asignación '=' crea un alias (dos nombres para una misma variable).
Si quieres copiar una lista usa la función `.copy()`

Variable Alias

- Si ambos elementos son listas, el operador de asignación crea un nuevo nombre a la variable
- Para crear una copia usa lista.copy()

```
1 L = [1, 2, 3]
```

```
2 C = L
```

```
3 L[0] = 99
```

```
4
```

```
5 print(L)
```

```
6 print(C)
```

Reemplazar por



```
2 C = L.copy()
```

Importante: El operador de asignación '=' crea un alias (dos nombres para una misma variable).

Si quieres copiar una lista usa la función .copy()

Ejemplos

Genera números aleatorios
dentro de un rango



```
1 from random import randrange
2 TRAJES = ['Picas', 'Diamantes', 'Treboles', 'Corazones']
3 VALORES = ['2', '3', '4', '5', '6', '7', '8', '9', '10',
4            'Jota', 'Reina', 'Rey', 'As']
5
6 valor = randrange(0, len(VALORES))
7 traje = randrange(0, len(TRAJES))
8 print(VALORES[valor], 'de', TRAJES[traje])
```

```
1 from random import randrange
2 TRAJES = ['Picas', 'Diamantes', 'Treboles', 'Corazones']
3 VALORES = ['2', '3', '4', '5', '6', '7', '8', '9', '10',
4            'Jota', 'Reina', 'Rey', 'As']
5
6 mazo = []
7 for traje in TRAJES:
8     for valor in VALORES:
9         mazo.append(valor + ' de ' + traje)
10
11 print(mazo)
```

Utilidades

Crear lista con valores de teclado	<pre>L = [] #lista vacía for i in range(N): v = int(input()) L.append(v)</pre>
Imprimir valores en lista (uno por uno)	<pre>for elem in L: print(elem) # alternativa for i in range(N): print(L[i])</pre>
Encontrar el máximo valor en una lista	<pre>maxi = L[0] for elem in L: if elem > maxi: maxi = elem print(maxi)</pre>
Encontrar el mínimo valor en una lista	<pre>mini = L[0] for elem in L: if elem < mini: mini = elem print(mini)</pre>

Obtener el promedio	<pre># promedio suma = 0.0 for elem in L: suma = suma + elem prom = suma/N</pre>
Copiar elementos a otra lista	<pre>L2 = [] for elem in L: L2.append(elem)</pre>
Crear nueva listas con elementos invertidos	<pre>N = len(L) R = [] for i in range(N): j = N-i-1 R.append(L[j])</pre>
Invertir elementos del arreglo	<pre>for i in range(N-1): temp = L[i] L[i] = L[N-i-1] L[N-i-1] = temp</pre>

Utilidades

	Código Típico	Función en Python
Encontrar el máximo valor en una lista	<pre>maxi = L[0] for elem in L: if elem > maxi: maxi = elem print(maxi)</pre>	<pre>maxi = max(L)</pre>
Encontrar el mínimo valor en una lista	<pre>mini = L[0] for elem in L: if elem < mini: mini = elem print(mini)</pre>	<pre>mini = min(L)</pre>
Obtener el promedio	<pre># promedio suma = 0.0 for elem in L: suma = suma + elem prom = suma/N</pre>	<pre>prom = sum(L)/len(L)</pre>
Copiar elementos a otra lista	<pre>L2 = [] for elem in L: L2.append(elem)</pre>	<pre>L2 = L.copy()</pre>
Invertir elementos del arreglo	<pre>for i in range(N): temp = L[i] L[i] = L[N-i-1] L[N-i-1] = temp</pre>	<pre>L = list(reversed(L))</pre>

L = [4, 8, 15, 16, 23, 42]

DRY: Don't repeat yourself!

Si tienes más de una forma de expresar un algoritmo, escoge la más sencilla.



DRY

* https://en.wikipedia.org/wiki/Don%27t_repeat_yourself

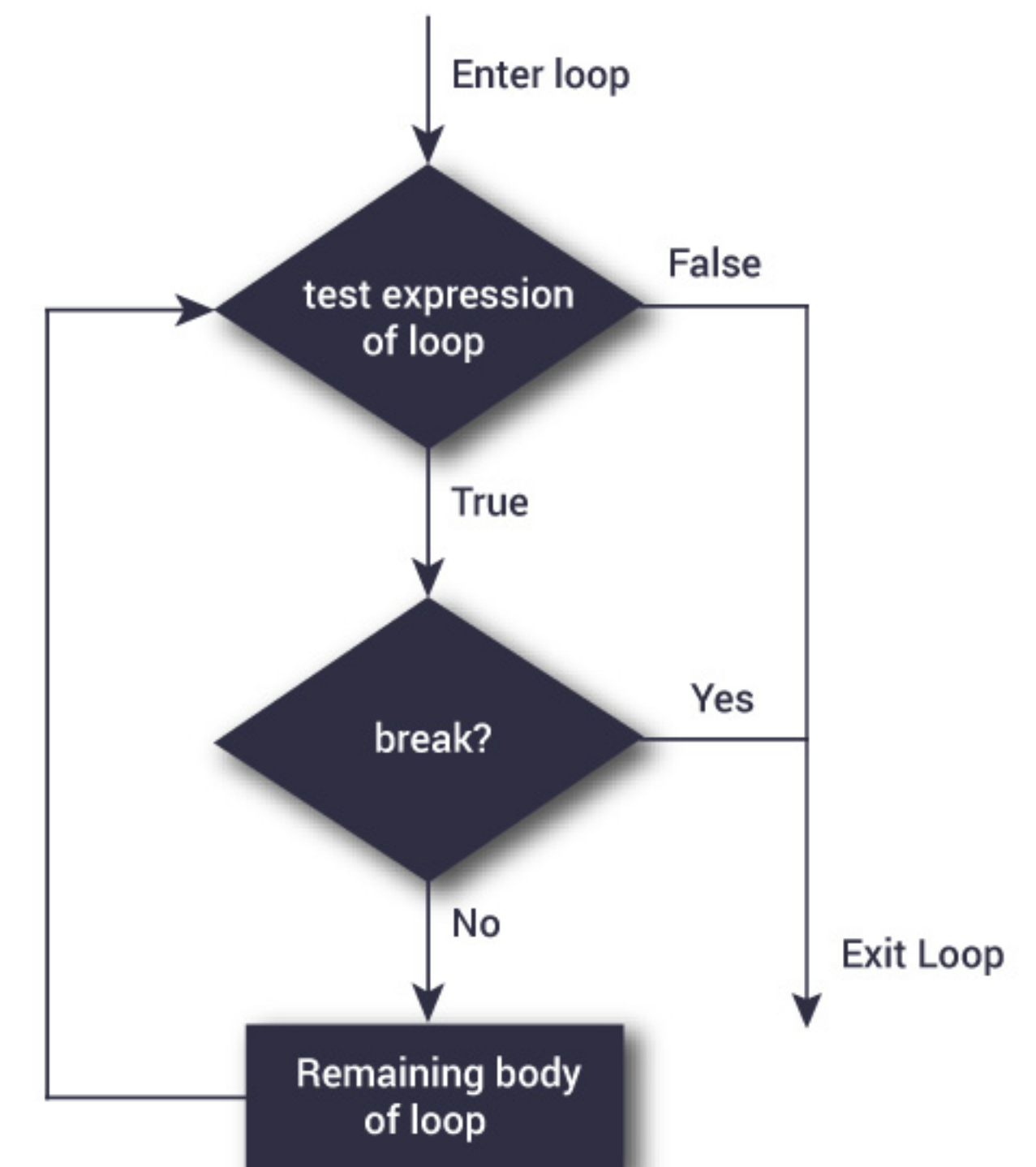
Ciclos II: **break**

- **break**: Sirve para detener ciclos antes de que se recorra una secuencia o la condición en **while** no se cumpla.
- **Ventaja**: podemos ahorrar tiempo de procesador (muuuuuy poco).
- **Desventaja**: código más complejo.

```
for var in secuencia:
    # código dentro del ciclo for
    if condicion:
        break # detiene el ciclo for
    # código dentro del ciclo for
#código fuera del ciclo for

--

while test expresión:
    # código dentro del ciclo while
    if condicion:
        break # detiene el ciclo while
    # código dentro del ciclo while
#código fuera del ciclo while
```



Ciclos II: **break**

```
L=[1,3,4,0,8]

for e in L:
    if e == 0:
        break
    print(e)
```

Nota: si necesitas usar **break**, verifica que sea la alternativa más sencilla.

¿Qué hacen los programas a, b, c y d?

a)

```
1 L = 1000000*[0, -1, 3, 5, 9, 10, 12, 99, 33]
2 print('len(L):', len(L))
3
4 a = False
5 for e in L:
6     if e < 0:
7         a = True
8         break
9 print(a)
```

b)

```
1 L = 1000000*[0, -1, 3, 5, 9, 10, 12, 99, 33]
2 print('len(L):', len(L))
3
4 a = False
5 for e in L:
6     if e < 0:
7         a = True
8 print(a)
```

c)

```
1 L = 1000000*[0, -1, 3, 5, 9, 10, 12, 99, 33]
2 print('len(L):', len(L))
3
4 a = False
5 i = 0
6 while i < len(L):
7     if L[i] < 0:
8         a = True
9         break
10    i += 1
11 print(a)
```

d)

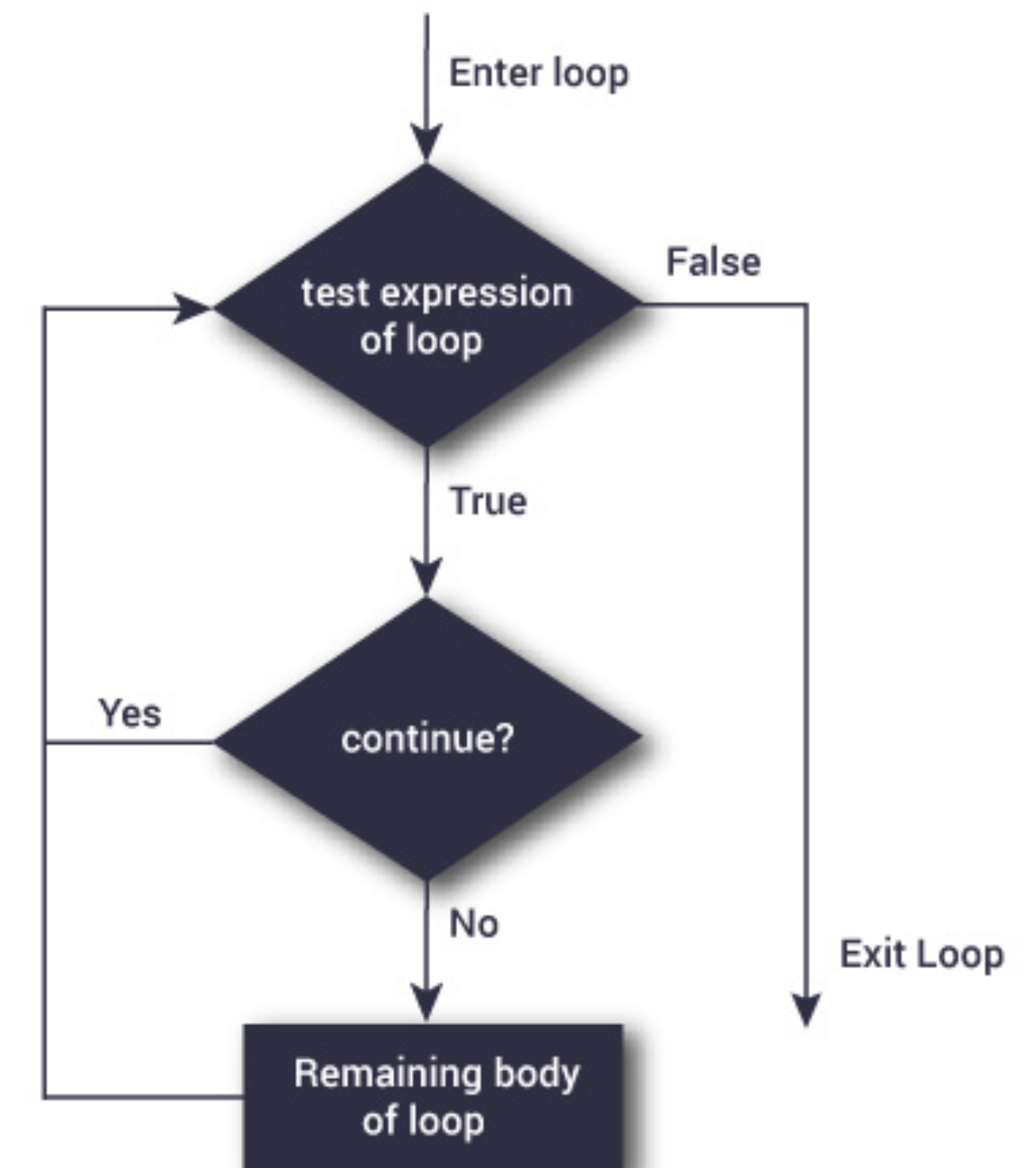
```
1 L = 1000000*[0, -1, 3, 5, 9, 10, 12, 99, 33]
2 print('len(L):', len(L))
3
4 a = False
5 i = 0
6 while i < len(L):
7     if L[i] < 0:
8         a = True
9     i += 1
10 print(a)
```

P: ¿Cuál de todos te gusta más? ¿Por qué?

Ciclos II: **continue**

- **continue**: Sirve para saltar alguna iteración (ej.: ignorar elementos negativos).
 - **Ventaja**: podemos ahorrar tiempo de procesador (muuuuuy poco).
 - **Desventaja**: código más complejo.

```
for var in secuencia:  
    # código dentro del ciclo for  
    if condicion:  
        continue # salta a siguiente iteración  
    # código dentro del ciclo for  
  
#código fuera del ciclo for  
  
while test expresión:  
    # código dentro del ciclo while  
    if condicion:  
        continue # salta a siguiente iteración  
    # código dentro del ciclo while  
  
#código fuera del ciclo while
```



Ciclos II: `continue`

```
L=[1,3,4,0,8]

for e in L:
    if e == 0:
        continue
    print(e)
```

Nota: si necesitas usar `continue`, verifica que sea la alternativa más sencilla.

¿Qué hacen los programas a, b, c y d?

a)

```
1 L = [0, -1, 3, 5, 9, 10, 12, 99, 33]
2 t = 0
3 for e in L:
4     if e < 0:
5         continue
6     t += e
7 print(t)
```

← Salta a siguiente iteración

b)

```
1 L = [0, -1, 3, 5, 9, 10, 12, 99, 33]
2 t = 0
3 for e in L:
4     if e >= 0:
5         t += e
6 print(t)
```

c)

```
1 L = [0, -1, 3, 5, 9, 10, 12, 99, 33]
2 t = 0
3 i = 0
4 while i < len(L)
5     e = L[i]
6     i += 1
7     if e < 0:
8         continue
9     t += e
10 print(t)
```

← Salta a siguiente iteración

d)

```
1 L = [0, -1, 3, 5, 9, 10, 12, 99, 33]
2 t = 0
3 i = 0
4 while i < len(L)
5     e = L[i]
6     i += 1
7     if e >= 0:
8         t += e
9 print(t)
```

P: ¿Cuál de todos te gusta más? ¿Por qué?

Resumen

Conceptos

- **Lista:** secuencia de elementos
- **String:** secuencia de caracteres (texto)
- **Alias:** nuevo nombre a una variable. Si modifico el contenido en una, se modifica en la otra también.
- **Continue:** saltar una iteración en ciclo while/for
- **Break:** detener un ciclo for/while

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

https://docs.python.org/3/reference/lexical_analysis.html

Funciones

- **len(lista):** tamaño de una lista o de un string
- **elem.copy():** crear copia de variable elem

		Built-in Functions		
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

<https://docs.python.org/3/library/functions.html>