Python 3

> {‘str’: [0]}

len()

Esta función built-in devuelve la longitud del objeto que se pase como parámetro.

Este valor es determinado utilizando el método `__len__()` del objeto.

`len(<obj>)`

```
>>> len(range(10))
10
>>> len([1, 2, 3])
3
>>> len('hello')
5
>>> len({1, 2, 3})
3
>>> len((1, 2, 3))
3
>>> len({'a': 0})
1
```

in / not in

El operador `in` devuelve `True` si un objeto pertenece a otro y `False` si no pertenece. El operador `not in` hace lo contrario.

```
>>> 2 in [1, 2, 3]
True
>>> None in [1, 2, 3]
False
>>> 2 in (1, 2, 3)
True
>>> [1] in ('a', 0, [1])
True
>>> 0 in {'a': 0}  # dicts use their keys to test membership
False
>>> 'a' in {'a': 0}  # dicts use their keys to test membership
True
>>> 'World' in 'Hello, World!'
True
>>> 'world' in 'Hello, World!'
False
```

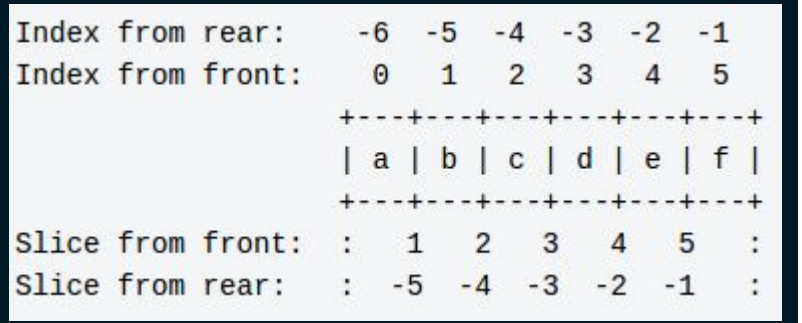
Slices (I)

Los slices se pueden utilizar para acceder a sub-secuencias de los tipos de datos que lo permitan. Algunos de estos tipos son: `list`, `tuple` y `str`.

Algunas formas de utilizar el operador slice son:

```
obj[start:end]
obj[start:end:step]
obj[start:]
obj[:end]
obj[:]
```

```
>>> l = list(range(10))
>>> t = ('a', 0, 'alpha', '1st')
>>> s = 'Hello World!'
>>> l[2:8]
[2, 3, 4, 5, 6, 7]
>>> l[2:]
[2, 3, 4, 5, 6, 7, 8, 9]
>>> l[:8]
[0, 1, 2, 3, 4, 5, 6, 7]
>>> l[2:8:2]
[2, 4, 6]
>>> l[0:-1]
[0, 1, 2, 3, 4, 5, 6, 7, 8]
>>> l[-3:-1]
[7, 8]
>>> l[::-1]
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
>>> t[1:3]
(0, 'alpha')
>>> s[:5]
'Hello'
>>> s[:100]
'Hello World!'
>>> s[5:1]
''
```



Slices (II)

Slices (III)

El operador slice se puede utilizar para copiar listas.

```
>>> l1 = [0, 1, 2, 3, 4]
>>> l2 = l1
>>> l2[0] = ':D'
>>> l1
[':D', 1, 2, 3, 4]
>>> l1 = [0, 1, 2, 3, 4]
>>> l2 = l1[:]  # l2 is a copy of l1
>>> l2[0] = ':D'
>>> l1
[0, 1, 2, 3, 4]
```

Slices (IV)

Cuando se está utilizando listas el operador slice puede ser utilizado del lado de la asignación.

```
>>> l = list(range(10))
>>> l
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> l[0:5] = [4, 3, 2, 1, 0]  # slices can be used to update a whole sublist at once
>>> l
[4, 3, 2, 1, 0, 5, 6, 7, 8, 9]
>>> l[0:5] = []  # slices can be used to remove elements from a list
>>> l
[5, 6, 7, 8, 9]
>>> l[2:2] = [-7, 0]  # slices can be used to insert elements to a list
>>> l
[5, 6, -7, 0, 7, 8, 9]
```

Listas

Metodos de lists

`list.append(x)`

`list.extend(iterable)`

`list.insert(i, x)`

`list.remove(x)`
Removes first encounter. Raises `ValueError` if `x` is not found.

`list.pop([i])`
Raises `IndexError`.

`list.clear()`

`list.index(x[, start[, end]])`
Raises `ValueError` if `x` is not found.

`list.count(x)`

`list.sort(key=None, reverse=False)`
In-place.

`list.reverse()`
In-place.

`list.copy()`
Shallow copy.

Eliminar elementos con del

La sentencia `del` se puede utilizar para eliminar elementos de una lista.

```
>>> a = list(range(10, 100, 5))
>>> a
[10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95]
>>> del a[3]  # remove an element given its index
>>> a
[10, 15, 20, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95]
>>> del a[1:5]  # remove a whole slice
>>> a
[10, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95]
>>> del a[:]  # clear the list
>>> a
[]
```

```python
l = [list(range(sl)) for sl in range(10)]  # create a list of lists

print(l)  # print the whole list

for sl in l:
    print(sl)  # print each sublist
    for v in sl:
        print(v)  # print each value

for i, sl in enumerate(l):
    for j, v in enumerate(sl):
        print(l[i][j] == sl[j] == v)  # always True
```

Recorriendo lists

Strings

Metodos de strings

`s.count(sub[, start[, end]])`

`s.endswith(suffix[, start[, end]])`

`s.find(sub[, start[, end]])`
Devuelve el índice o -1 si el resultado es negativo.

`s.join(iterable)`

`s.lower()`

`s.split(sep=None, maxsplit=-1)`

`s.splitlines([keepends])`

`s.startswith(prefix[, start[, end]])`

`s.strip([chars])`

`s.upper()`

Comparando strings

Cuando se comparan datos del tipo string se utiliza el valor Unicode de cada carácter. Esto implica que estas comparaciones son case-sensitive.

```
>>> 'hello' == 'bye'
False
>>> 'hello' == 'hello'
True
>>> 'hello' == 'HELLO'
False
>>> 'banana' > 'anana'
True
>>> 'Banana' > 'anana'
False
```

f”Yeah!”

Desde Python 3.6 están disponibles las f-strings, una herramienta para darle formato a strings.

En una f-string se evalúan las expresiones que estén contenidas dentro de llaves {} y luego se utiliza su resultado para componer el texto con formato.

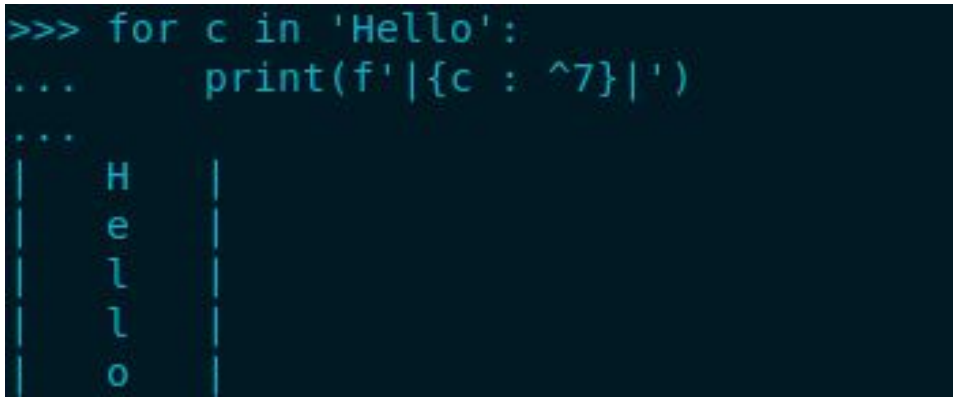
Se puede usar : para customizar el formato utilizado.

```
>>> print(f'ten plus five equals {10 + 5}')
ten plus five equals 15
>>> print(f'|{10 ** 3 : >8}|')
|    1000|
>>> print(f'|{10 ** 3 : <8}|')
|1000    |
>>> print(f'|{10 ** 3 : ^8}|')
|  1000  |
>>> print(f'{10 / 3 : .2f}')
 3.33
>>> print(f'{datetime.now() : %m/%d -- %H:%M}')
 12/09 -- 14:37
```

```
>>> # in order to make a brace appear in your
... # string, you must use double braces
...
>>> print(f'{{ 10 + 5 }}')
{ 10 + 5 }
>>> print(f'{{ {10 + 5} }}')
{ 15 }
```

Recorriendo strings

Un dato de tipo string se itera sobre los caracteres que lo componen.



Diccionarios

Metodos de dicts

`d.clear()`

`d.copy()`
Shallow copy.

`d.get(key[, default])`
Devuelve `None` sí default no es especificado.

`d.items()`
Devuelve una vista de los pares (key, value).

`d.keys()`
Devuelve una vista de las keys.

`d.pop(key[, default])`
Elimina el valor y lo retorna. Sí `default` no es especificado y `key` no existe se lanza un `KeyError`.

`d.values()`
Devuelve una vista de los values.

```python
d = {i: chr(i) for i in range(65, 70)}  # {65: 'A', 66: 'B', ...}

print(d)
print('-' * len(str(d)))

# dicts are iterated over their keys
for key in d:
    d[key] = d[key].lower()  # convert the letters to lower case

# use items() to iterate over the values of a dict
for key, value in d.items():
    print(f'{key} => {value}')
```

```
{65: 'A', 66: 'B', 67: 'C', 68: 'D', 69: 'E'}
----------------------------------------------
65 => a
66 => b
67 => c
68 => d
69 => e
```

Recorriendo dicts

code()

Ejercicios 3

- Escribir un programa que determine si una palabra es un palíndromo.
- Escribir un programa que dada una cadena de texto obtenga y muestre la siguiente información:
    - La cantidad de palabras de todo el texto.
    - La palabra más larga y su cantidad de letras.
    - La cantidad de veces que se utiliza cada palabra.

  La comparación de texto no debe discriminar mayusculas de minusculas.

Bibliografía

- Python docs
- How to Think Like a Computer Scientist: Learning with Python
- Python 3's f-Strings: An Improved String Formatting Syntax