`while` loop

Code runs while a condition holds True.

In [1]:
```
i = 10
j = 0
while (j<5) or (i>6):
    print(f"i={i:<2}, j={j}")
    j += 1
    i -= 1 # i = i -1
```

```
i=10, j=0
i=9 , j=1
i=8 , j=2
i=7 , j=3
i=6 , j=4
```

note: beware of `while` loops that never come to an end!

Operators `any` and `all` are used to process iterables of boolean values:

- `any` returns `True` when at least one value is `True`
- `all` returns `True` when all values are `True`

note: if `all` returns `True` then `any` returns also `True`.

In [2]:
```
data = [1, 2, 3, 4, 5]
data_cond = [e > 3 for e in data]
print(any(data_cond))
print(all(data_cond))
```

```
True
False
```

In [3]:
```
data = [1, 2, 3, 4, 5]
data_cond = [e > 0 for e in data]
print(any(data_cond))
print(all(data_cond))
```

```
True
True
```

`for` loop

`for` makes it possible to go through the values of any **iterable** object.

Lists

In [4]:
```
for k in [0, 1, 2, 3, 4]:
    print(k)
```

```
0
1
2
3
4
```

Generators-like

In [5]:
```
for k in range(5):
    print(k)
```

```
0
1
2
3
4
```

In [6]:
```
gen = (k//2 for k in range(0, 10, 2))
for k in gen:
    print(k)
```

```
0
1
2
3
4
```

Dictionaries

Iteration is done on the keys of the dictionary:

In [7]:
```
dic = {"key1": "value1", "key2": "value2"}
for k in dic:
    print(k)
```

```
key1
key2
```

If both keys and values are needed, one must use the `items()` method:

In [8]:
```
dic = {"key1": "value1", "key2": "value2"}
for k, v in dic.items():
    print(k, v)
```

```
key1 value1
key2 value2
```

Indexes and values

The `enumerate` function applies to any object that can be iterated over. It returns both the index, and the value. In Python, **the first index is always 0**.

In [9]:
```
for idx, k in enumerate(["A", "B", "C"]):
    print(idx, k)
```

```
0 A
1 B
2 C
```

Group using `zip`

`zip` makes tuples by extracting an element from each iterable it is given, as long as at least one iterable has no more elements.

In [10]:
```
iter_1 = [1,2,3,4]      # longueur: 4
iter_2 = "abcdefgh"     # longueur: 8
for i, j in zip(iter_1, iter_2):
    print(i, j)
```

```
1 a
2 b
3 c
4 d
```

`break`

Get out of a control flow structure:

In [11]:
```
for k in range(5):
    print(f"k={k}")
    if k == 3:
        break
```

```
k=0
k=1
k=2
k=3
```

If multiple control flow structure are ensted, `break` only exits the deepest level (innermost):

In [12]:
```
j = 0
while (j < 5):
    print(f"\nj={j}", end="")
    for k in range(5):
        print(f", k={k}", end="")
        if k == 3:
            break
    j += 1
```

```
j=0, k=0, k=1, k=2, k=3
j=1, k=0, k=1, k=2, k=3
j=2, k=0, k=1, k=2, k=3
j=3, k=0, k=1, k=2, k=3
j=4, k=0, k=1, k=2, k=3
```

`continue`

Interrupt current iteration and go the next one.

In [13]:
```
j = 0
while j < 5:
    j += 1
    if (j%3 == 0):
        continue
    print(f"j={j}")
```

```
j=1
j=2
j=4
j=5
```

`else` statement

The content of `else` is ran only and only if the previously ran control flow structure never met a `break`.

In [14]:
```
for k in range(5):
    if k > 15:
        break
else:
    print("Values lower than 15")
```

```
Values lower than 15
```

`if` conditions

Keywords are:

- `if` : test whether a condition holds `True` , independantly from previous tests
- `elif` : (optional) test whether a condition holds `True` if and only if previous `if` and `elif` did not hold `True`
- `else` : (optional) code that mus

In [15]:
```
word = "abracadra"
if "x" in word:
    print("'x' in word")
if "a" in word:
    print("'a' in word")
if "y" in word:
    print("'y' in word")
elif len(word) < 5:
    print("Small word")
else:
    print("Fallback to 'else'")
```

```
'a' in word
Fallback to 'else'
```