Informàtica
Curs 2023-2024

Iteradors – part 2

D.Tost

El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

Resum definicions

Objecte contenidor: objecte que fa referència a d'altres objectes, per exemple tuples, llistes, diccionaris, strings. Està format per diversos “elements”.

Objecte iterable: objecte contenidor que disposa d’un mecanisme per lliurar els seus elements d'un en un (e.g. list, tuple, dict, str, set)

Iterador: Objecte capaç d’iterar sobre una seqüencia d’elements. Els iteradors són iterables però no tots els objectes iterables són iteradors (e.g una llista)

La funció `next(it)`: Permet d’obtenir el següent element d’un iterador it.

La funció `iter(obj)`: Retorna un iterador de l’objecte iterable obj. Si obj és un iterador, retorna obj.

Funcions generadores: creen iteradors utilitzant la instrucció **`yield`**. Cada vegada que es crida la funció **`next`** sobre l’iterador creat s’executen totes les instruccions de la funció fins al següent **`yield`** inclòs.

El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

Funcions built-in

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

abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	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()	
delattr()	hash()	memoryview()	set()	

Les marcades o bé:

- tenen **iterables** com a paràmetres (max, min, sum, all, any...)
- o retornen **objectes iterables** (dict, list, tuple, range, set)
- o retornen **iteradors** (map, filter, zip)

El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

Funcions builtins que tenen iterables com a paràmetre (però no són generadores) Exemple: sum

sum(*iterable*[, *start*])

Sums *start* and the items of an *iterable* from left to right and returns the total. *start* defaults to `0`. The *iterable*'s items are normally numbers, and the start value is not allowed to be a string.

```
>>> sum((1, 2, 3))
6
>>> def it() :
...         yield 4
...         yield 5
...         yield 6
...
>>> a = it()
>>> sum(a)
15
```

(1, 2, 3) és un tuple, és iterable com les classes `list, tuple, set, dict`

`it` és una funció generadora que crea iteradors sobre la seqüència 4, 5, 6.

`a` és un iterador, per tant és iterable.

sum és una funció built-in que suma un darrera de l'altra els elements de l'iterable que rep com a paràmetre

Alerta! No és una funció generadora! L'expliquem per a il·lustrar funcions que tenen iterables com a paràmetres

El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

Funcions builtins que tenen paràmetres iterables

all(*iterable*)

Return `True` if all elements of the *iterable* are true (or if the iterable is empty).

```
>>> all([2+2== 4, True, isinstance(2, int)])
True
```

any(*iterable*)

Return `True` if any element of the *iterable* is true. If the iterable is empty, return `False`.

```
>>> any([2+2== 3, False, isinstance(2, int)])
True
```

Alerta! No són funcions generadores! Les expliquem per a il·lustrar funcions que treballen amb iterables. Exploreu-les!

El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

Funcions que retornen iteradors
Funció map

map(*function*, *iterable*, ...)

Return an iterator that applies *function* to every item of *iterable*, yielding the results. If additional *iterable* arguments are passed, *function* must take that many arguments and is applied to the items from all iterables in parallel. With multiple iterables, the iterator stops when the shortest iterable is exhausted. For cases where the function inputs are already arranged into argument tuples, see `itertools.starmap()`.

```
>>> def restaun(n) :
...     return n-1
...
>>> it = map(restaun, [3, 7, 11])
>>> list(it)
[2, 6, 10]
```

`restaun` és una funció que rep un enter i retorna un enter

`it` és un iterador retornat per la funció map
`it` lliura els elements retornats per la funció `restaun` per cada element de l'iterable [3, 7, 11].

`list` retorna la llista dels elements de l'iterable
Què passaria si féssim `next(it)` després de crear la llista?

```
>>>>>> def escapça(a):
...     return a[1:-1]
...
>>> escapça('hola')
'ol'
>>> it = map(escapça, ['hola', 'ràbia', 'calçot',
'mama', 'papa', 'avi'])
>>> list(it)
['ol', 'àb', 'am', 'ap', 'v']
```

`it` lliura cadascun dels strings de la llista un cop escapçats

El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

Funcions que retornen iteradors
Funció map

Dissenyeu una funció que donat un **iterable** sobre una seqüència d'enters retorna un **iterador** dels elements de l'iterable havent-los multiplicat per 2 i restat 6

```
>>> def f(x) :
...     return 2*x -6
...
>>> it = map(f, [2, 5, 8])
>>> list(it)
[-2, 4, 10]
```

Dissenyeu una funció que donats dos **iterables** sobre seqüències d'enters retorna un **iterador** de la suma 1 a 1 dels elements dels dos iterables.

```
>>> def f(x, y) :
...      return x + y
...
>>> it = map( f, [0, 0, 2, 4, 8], [4, 5, 8, -1,
9])
>>> list(it)
[4, 5, 10, 3, 17]
>>> it = map( f, [20, 10, 2, 4, 8], [4, 5])
>>> list(it)
[24, 15]
```

El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

Funcions anònimes
Expressions lambda

Es poden crear funcions anònimes quan es necessiten puntualment i no s'han de reutilitzar. https://docs.python.org/3/reference/expressions.html#lambda

Tenen la forma: `lambda paràmetres: expressió`
Que seria equivalent a quelcom com:
```
def nom(paràmetres) :
    return expressió
```

Només poden retornar una expressió, **no fer un bloc de sentències.**

```
>>> it = map(lambda x: 2*x -6, [2, 5, 8])
>>> list(it)
[-2, 4, 10]

>>> it = map(lambda x, y: x + y, [2, 4, 8], [4, 5, 8, -1, 9])
>>> list(it)
[6, 9, 16]
```

Avançat!!
Per la vostra curiositat

Es poden utilitzar *list comprehensions* per a que facin coses més complexes.

```
>>> it3 = map(lambda x: x if x%2==0 else -1, [2, 3, 4, 5, 7, 8, 10])
>>> list(it3)
[2, -1, 4, -1, -1, 8, 10]
```

El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

Funcions que retornen iteradors
Funció map

Dissenyeu una funció que donat un enter i un **iterable** sobre una seqüència d'enters retorna un **iterador** dels elements de l'iterable havent-lis restat l'enter donat.

Voldríem fer quelcom com això, però la funció resta que es passa a map només pot rebre com a paràmetre l'element de l'iterable no `n`!!!

??
??

```
>>> def resta(n, x) :
...     return n-x
...
>>> def f(n, it):
...     return map(resta, it)
```

Solució: utilitzar una funció anònimes (lambda)

```
>>> def f(n, it) :
...     return map(lambda x: x-n, it)
...
>>> list(f(3, [2, 5, 8]))
[-1, 2, 5]
```

El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

Funcions que retornen iteradors
Funció map

Dissenyeu una funció que donat un enter i un **iterable** sobre una seqüència d'enters retorna un **iterador** dels elements de l'iterable havent-lis restat l'enter donat.

Voldríem fer quelcom com això, però la funció resta que es passa a map només pot rebre com a paràmetre l'element de l'iterable no `n`!!!

??
??

```
>>> def resta(n, x) :
...     return n-x
...
>>> def f(n, it):
...     return map(resta, it)
```

Solució alternativa (nivell avançat): utilitzar una funció dins de la funció

```
>>> def f(n, it):
...     def resta(x) :
...         return x-n
...     return map(resta, it)
...
>>> list(f(3, [2, 5, 8]))
[-1, 2, 5]
```

El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

Funcions que retornen un iterador
funció filter

filter(*function*, *iterable*)

Construct an iterator from those elements of *iterable* for which *function* returns true. *iterable* may be either a sequence, a container which supports iteration, or an iterator. If *function* is `None`, the identity function is assumed, that is, all elements of *iterable* that are false are removed.

```
>>> def f(x) :
...     return x % 2 == 0
...
>>> it = filter(f, [2, 3, 4, 5, 9])
>>> list(it)
[2, 4]
>>> it = filter(lambda x: x%2 == 0, [2, 3, 4, 5, 9])
>>> list(it)
[2, 4]
>>> it = filter(lambda x: x%2 == 0 and 0<=x<=3, [2, 3, 4, 5, 9])
>>> list(it)
[2]
>>> it = filter(lambda x, y: x >y, [2, 3, 4, 5, 9], [3, 2, -3])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: filter expected 2 arguments, got 3


>>> list(filter(lambda s: s[0] == 'a', ['arbre', 'tren', 'camió', 'amor']))
['arbre', 'amor']
```

Fa `yield` només dels múltiples de 2

També suport les funcions lambda

Una expressió complexa

Només accepta un iterable! Després veurem com resoldre-ho

El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

Funcions que retornen un iterador funció zip

zip(**iterables*)

Make an iterator that aggregates elements from each of the iterables.

```
>>> it = zip([41, 24, 35], ('alice', 'pere', 'joan'))
>>> list(it)
[(41, 'alice'), (24, 'pere'), (35, 'joan')]
>>> list(zip([41, 24, 35], ('alice', 'pere', 'joan', 'ernest')))
[(41, 'alice'), (24, 'pere'), (35, 'joan')]
>>> list(zip([41, 24, 35], ('alice', 'pere', 'joan', 'ernest'), [True,
False]))
[(41, 'alice', True), (24, 'pere', False)]
```

Sabríeu ara resoldre el problema de la transparència anterior?

```
>>> it = filter(lambda x, y: x >y, [2, 3, 4, 5, 9], [3, 2, -3])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: filter expected 2 arguments, got 3
```

Proveu: (hi ha d'altres de maneres de fer-ho)

```
>>> it = map(lambda x: x[0], filter(lambda t: t[0]> t[1], zip([2, 3, 4, 5, 9], [3, 2, -3])))
>>> list(it)
[3, 4]
```

El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

itertools

Un mòdul que proporciona funcions convenients per a crear iteradors

Consulteu la documentació. Alguns iteradors són infinits.

Infinite Iterators:

Iterator	Arguments	Results	Example
`count()`	start, [step]	start, start+step, start+2*step, ...	`count(10) --> 10 11 12 13 14 ...`
`cycle()`	p	p0, p1, ... plast, p0, p1, ...	`cycle('ABCD') --> A B C D A B C D ...`
`repeat()`	elem [,n]	elem, elem, elem, ... endlessly or up to n times	`repeat(10, 3) --> 10 10 10`

```
>>> it = itertools.count(56, 4)
>>> for i in  range(5):
...     print(next(it), end= '-')
...
56-60-64-68-72-
```

Genera els elements
a partir de 56 sumant 4

https://docs.python.org/3/library/itertools.html?highlight=itertools#module-itertools

El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

Itertools- Exemple

Dissenyeu una funció que donat un iterable retorna un iterador dels elements d'aquest iterable restant-lis un valor donat `n`

Una solució diferent a l'anterior:

```
>>> def f(n, it):
...     return map(lambda x:x[0]-x[1],
zip(itertools.repeat(n), it))
...
>>> list(f(3, [3, 4, 5]))
[0, -1, -2]
```

x[0] és sempre n
x[1] té valor successiu cadascun dels elements de l'iterable

Alerta, repeat és infinit, però it és un iterable (finit). Quan s'esgoten els seus elements, zip s'atura.

https://docs.python.org/3/library/itertools.html?highlight=itertools#module-itertools

El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

itertools

https://docs.python.org/3/library/itertools.html?highlight=itertools#module-itertools

Iterator	Arguments	Results	Example
`accumulate()`	p [,func]	p0, p0+p1, p0+p1+p2, ...	`accumulate([1,2,3,4,5]) --> 1 3 6 10 15`
`chain()`	p, q, ...	p0, p1, ... plast, q0, q1, ...	`chain('ABC', 'DEF') --> A B C D E F`
`chain.from_iterable()`	iterable	p0, p1, ... plast, q0, q1, ...	`chain.from_iterable(['ABC', 'DEF']) --> A B C D E F`
`compress()`	data, selectors	(d[0] if s[0]), (d[1] if s[1]), ...	`compress('ABCDEF', [1,0,1,0,1,1]) --> A C E F`
`dropwhile()`	pred, seq	seq[n], seq[n+1], starting when pred fails	`dropwhile(lambda x: x<5, [1,4,6,4,1]) --> 6 4 1`
`filterfalse()`	pred, seq	elements of seq where pred(elem) is false	`filterfalse(lambda x: x%2, range(10)) --> 0 2 4 6 8`
`groupby()`	iterable[, keyfunc]	sub-iterators grouped by value of keyfunc(v)	
`islice()`	seq, [start,] stop [, step]	elements from seq[start:stop:step]	`islice('ABCDEFG', 2, None) --> C D E F G`
`starmap()`	func, seq	func(*seq[0]), func(*seq[1]), ...	`starmap(pow, [(2,5), (3,2), (10,3)]) --> 32 9 1000`
`takewhile()`	pred, seq	seq[0], seq[1], until pred fails	`takewhile(lambda x: x<5, [1,4,6,4,1]) --> 1 4`
`tee()`	it, n	it1, it2, ... itn splits one iterator into n	
`zip_longest()`	p, q, ...	(p[0], q[0]), (p[1], q[1]), ...	`zip_longest('ABCD', 'xy', fillvalue='-') --> Ax By C- D-`

El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

Itertools: chain i chain.from_iterable

`itertools.chain(*iterables)` ← 1 o més iterables

Make an iterator that returns elements from the first iterable until it is exhausted, then proceeds to the next iterable, until all of the iterables are exhausted. Used for treating consecutive sequences as a single sequence.

```
>>> it = chain([1, 2], 'ABC', iter((2, 'A')))
>>> next(it)
1
>>> next(it)
2
>>> next(it)
'A'
>>> next(it)
'B'
>>> next(it)
'C'
>>> next(it)
2
>>> next(it)
'A'
>>> next(it)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

`chain` lliura tots els elements del primer iterable, després els del segon i així successivament, «encadena» els iterables.

El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

Itertools: chain i chain.from_iterable

classmethod chain.**from_iterable**(*iterable*) ← 1 sol iterable!

Alternate constructor for `chain()`. Gets chained inputs from a single iterable argument that is evaluated lazily.

```
>>> it = chain.from_iterable(['ABC', (1, 2), ['AA', 'BB']])
>>> next(it)
'A'
>>> next(it)
'B'
>>> next(it)
'C'
>>> next(it)
1
>>> next(it)
2
>>> next(it)
'AA'
>>> next(it)
'BB'
>>> next(it)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

`Chain.from_iterable` lliura tots els elements del primer element de l'iterable ('ABC'), després els del segon i així successivament.
Només un nivell de profunditat: no aplana!
e.g lliura 'AA' i 'BB' no 'A', 'A', 'B', 'B'

El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

Itertools: accumulate i compress

https://docs.python.org/3/library/itertools.html?highlight=itertools#module-itertools

```
>>> from itertools import accumulate
>>> list(accumulate(range(10)))
[0, 1, 3, 6, 10, 15, 21, 28, 36, 45]
```

Per cada element d'un iterable `accumula` lliura la suma dels elements fins a aquest element. Aquí:
0, 0+1, 0+1+2, 0+1+2+3, 0+1+2+3+4, ...

```
>>> list(itertools.compress(['alice', 'albert', 'pere',
'andreu'], [True, False, True, True]))
['alice', 'pere', 'andreu']
```

`compress` lliura tots els elements del primer iterable que corresponen a un valor `True` en el segon iterable.

El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

Itertools: tee, takewhile i dropwhile

https://docs.python.org/3/library/itertools.html?highlight=itertools#module-itertools

```
>>> it1, it2, it3 =itertools.tee([('Alice', 41), ('Joan', 23),
('Pere', 35)], 3)
>>> list(it1)
[('Alice', 41), ('Joan', 23), ('Pere', 35)]
>>> next(it1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
>>> next(it2)
('Alice', 41)
>>> next(it3)
('Alice', 41)
```

`tee` és útil quan volem operar dues o més vegades sobre un iterador: el copiem.

`dropwhile` no lliura cap element fins arribar al primer que compleix una condició. A partir d'aquest, els lliura tots.

```
>>> list(itertools.dropwhile(lambda x: x[0] == 'a', ['alice',
'albert', 'pere', 'andreu']))
['pere', 'andreu']
>>> list(itertools.takewhile(lambda x: x[0] == 'a', ['alice',
'albert', 'pere', 'andreu']))
['alice', 'albert']
```

`takewhile` lliura tots els elements fins fins arribar al primer que compleix una condició. A partir d'aquest, no lliura res.

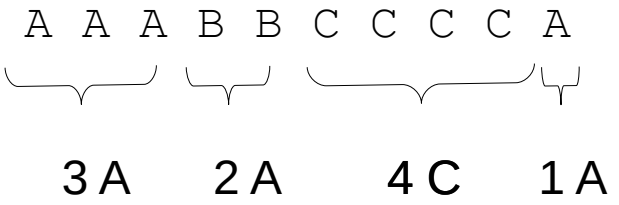
El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

Itertools groupby

> `itertools.`**`groupby`**(*`iterable`*, *`key=None`*)
>
> Make an iterator that returns consecutive keys and groups from the *iterable*. The *key* is a function computing a key value for each element. If not specified or is `None`, *key* defaults to an identity function and returns the element unchanged. Generally, the iterable needs to already be sorted on the same key function.

Crea un iterador de tuples: el valor del primer element de l'iterable i un iterador dels valors consecutius de l'iterable que són iguals a aquest (inclós l'element), i així successivament a partir del següent element diferent al primer.

```
>>> it = groupby('AAABBCCCCA')
>>> valor, it1 = next(it)
>>> valor, list(it3)
('A', ['A', 'A', 'A'])
>>> valor, it1 = next(it)
>>> valor, list(it1)
('B', ['B', 'B'])
>>> valor, it1 = next(it)
>>> valor, list(it1)
('C', ['C', 'C', 'C', 'C'])
>>> valor, it3 = next(it)
>>> valor, list(it1)
('A', ['A'])
```



El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

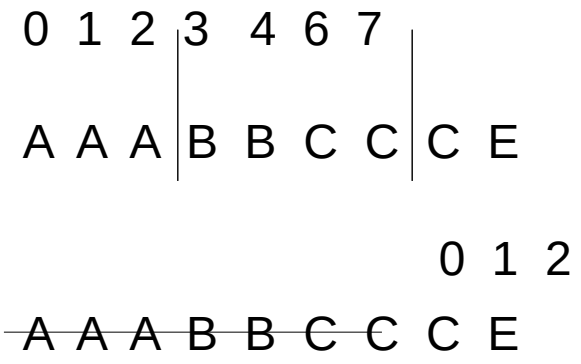
Itertools islice

itertools.**islice**(*iterable*, *stop*)
itertools.**islice**(*iterable*, *start*, *stop*[, *step*])

Make an iterator that returns selected elements from the iterable. If *start* is non-zero, then elements from the iterable are skipped until start is reached. Afterward, elements are returned consecutively unless *step* is set higher than one which results in items being skipped. If *stop* is `None`, then iteration continues until the iterator is exhausted, if at all; otherwise, it stops at the specified position. Unlike regular slicing, `islice()` does not support negative values for *start*, *stop*, or *step*.

Crea un iterador sobre una llesca de l'iterador inicial

```
>>> from itertools import islice
>>> it = iter(list('AAABBCCCE'))
>>> it2 = islice(it, 3, 7)
>>> list(it2)
['B', 'B', 'C', 'C']
>>> it2 = islice(it, 3, 7)
>>> list(it2)
[]
>>> next(it2)
Traceback (most recent call last):
  File "<stdin>", line 1, in
<module>
StopIteration
```



L'iterador ja només és sobre la Subseqüència C E.
La slice de 3 a a 7 és buida

El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

Itertools combinatòria

https://docs.python.org/3/library/itertools.html?highlight=itertools#module-itertools

Iterator	Arguments	Results
product()	p, q, ... [repeat=1]	cartesian product, equivalent to a nested for-loop
permutations()	p[, r]	r-length tuples, all possible orderings, no repeated elements
combinations()	p, r	r-length tuples, in sorted order, no repeated elements
combinations_with_replacement()	p, r	r-length tuples, in sorted order, with repeated elements

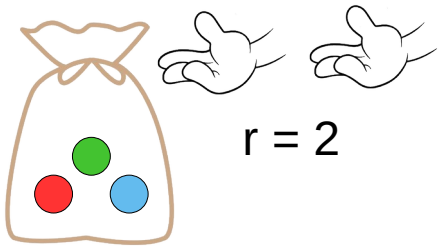
```
>>> list(itertools.product(['cara', 'creu'], repeat = 3))
[('cara', 'cara', 'cara'), ('cara', 'cara', 'creu'), ('cara', 'creu', 'cara'), ('cara', 'creu', 'creu'),
('creu', 'cara', 'cara'), ('creu', 'cara', 'creu'), ('creu', 'creu', 'cara'), ('creu', 'creu', 'creu')]

>>> list(itertools.product(['alt', 'baix'], ['guapo', 'lleig']))
[('alt', 'guapo'), ('alt', 'lleig'), ('baix', 'guapo'), ('baix', 'lleig')]
```

El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

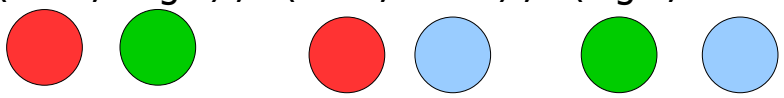
Itertools combinatòria

https://docs.python.org/3/library/itertools.html?highlight=itertools#module-itertools




```
>>> list(itertools.permutations(['r', 'g', 'b'], r=2))
[('r', 'g'), ('r', 'b'), ('g', 'r'), ('g', 'b'), ('b', 'r'), ('b', 'g')]
```

```
>>> list(itertools.combinations(['r', 'g', 'b'], r=2))
[('r', 'g'), ('r', 'b'), ('g', 'b')]
```



```
>>> list(itertools.combinations_with_replacement(['r', 'g', 'b'], r=2))
[('r', 'r'), ('r', 'g'), ('r', 'b'), ('g', 'g'), ('g', 'b'), ('b', 'b')]
```

El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.