08d-Saving-time-and-memory

September 16, 2022

1 Saving time and memory

There are some method that allow to save time, memory, coding effort and improve readability. Let us see some of them.

1.1 `map` function

`map(function, iterable, ...)` returns 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.

So, using a lambda function `map` can be used for instance in the following way.

[1]:
```
map(lambda a: (a, ),  range(3))
```

[1]:
```
<map at 0x1d5d6a0bac0>
```

Function `map` yelds an iterator object which can be latter consumed. To see its values its is enough to wrap in a list

[2]:
```
list(map(lambda a: (a, ), range(3)))
```

[2]:
```
[(0,), (1,), (2,)]
```

With multiple iterators

[3]:
```
list(map(lambda *a: a, range(3), 'abc', range(4, 7)))
```

[3]:
```
[(0, 'a', 4), (1, 'b', 5), (2, 'c', 6)]
```

1.2 `zip` function

`zip(*iterables)` returns an iterator of tuples, where the i-th tuple contains the i-th element from each of the argument sequences or iterables. The iterator stops when the shortest input iterable is exhausted. With a single iterable argument, it returns an iterator of 1-tuples. With no arguments, it returns an empty iterator.

[4]:
```
day_temperature = [18, 23, 30, 27, 15, 9, 22]
avg_temperature = [22, 21, 29, 24, 18, 18, 24]
```

```
list(zip(day_temperature, avg_temperature))
```

[4]:
```
[(18, 22), (23, 21), (30, 29), (27, 24), (15, 18), (9, 18), (22, 24)]
```

Of course, in some cases, we can use `map` is a somehow equivalent solution

[5]:
```
list(map(lambda *a: a, day_temperature, avg_temperature))
```

[5]:
```
[(18, 22), (23, 21), (30, 29), (27, 24), (15, 18), (9, 18), (22, 24)]
```

[6]:
```
for tt, at in zip(day_temperature, avg_temperature):
    print(f"Day's temperature was {tt}ºC being usual to have {at}ºC")
```

```
Day's temperature was 18ºC being usual to have 22ºC
Day's temperature was 23ºC being usual to have 21ºC
Day's temperature was 30ºC being usual to have 29ºC
Day's temperature was 27ºC being usual to have 24ºC
Day's temperature was 15ºC being usual to have 18ºC
Day's temperature was 9ºC being usual to have 18ºC
Day's temperature was 22ºC being usual to have 24ºC
```

`filter` function `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.

[7]:
```
test = [2, 5, 8, 0, 0, 1, 0]
```

[8]:
```
list(filter(None, test))
```

[8]:
```
[2, 5, 8, 1]
```

[9]:
```
list(filter(lambda x: x, test))
```

[9]:
```
[2, 5, 8, 1]
```

Keep only items > 4

[10]:
```
list(filter(lambda x: x > 4, test))
```

[10]:
```
[5, 8]
```

Of course it's not mandatory to use lambda function

[11]:
```
def square(x):
    return x**2

list(map(square, [1,2,3,4]))
```

[11]:
```
[1, 4, 9, 16]
```

[12]:
```
def high_speed(x):
    return x>120

list(filter(high_speed, [110,90,140,60]))
```

[12]:
```
[140]
```

1.3 List and dictionaries by Comprehensions

lets show it by examples ### lists

[13]:
```
[n ** 2 for n in range(10)]
```

[13]:
```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

[14]:
```
 [n ** 2 for n in range(10) if n % 2]
```

[14]:
```
[1, 9, 25, 49, 81]
```

[15]:
```
items = 'ABCD'
pairs = [(items[a], items[b])
         for a in range(len(items))
         for b in range(a, len(items))]
pairs
```

[15]:
```
[('A', 'A'),
 ('A', 'B'),
 ('A', 'C'),
 ('A', 'D'),
 ('B', 'B'),
 ('B', 'C'),
 ('B', 'D'),
 ('C', 'C'),
 ('C', 'D'),
 ('D', 'D')]
```

1.3.1 dictionaries

[16]:
```
from string import ascii_lowercase
letter_map = dict((c, k) for k, c in enumerate(ascii_lowercase, 1))
letter_map
```

[16]:
```
{'a': 1,
 'b': 2,
 'c': 3,
 'd': 4,
 'e': 5,
 'f': 6,
```

```
 'g': 7,
 'h': 8,
 'i': 9,
 'j': 10,
 'k': 11,
 'l': 12,
 'm': 13,
 'n': 14,
 'o': 15,
 'p': 16,
 'q': 17,
 'r': 18,
 's': 19,
 't': 20,
 'u': 21,
 'v': 22,
 'w': 23,
 'x': 24,
 'y': 25,
 'z': 26}
```

[17]:
```
word = 'Hello'
positions = {c: k for k, c in enumerate(word)}
positions
```

[17]:
```
{'H': 0, 'e': 1, 'l': 3, 'o': 4}
```

[18]:
```
{k: c for k, c in enumerate(word)}
```

[18]:
```
{0: 'H', 1: 'e', 2: 'l', 3: 'l', 4: 'o'}
```

1.3.2 Sets

[19]:
```
word = 'Hello'
set(c for c in word)
```

[19]:
```
{'H', 'e', 'l', 'o'}
```

[20]:
```
{c for c in word}
```

[20]:
```
{'H', 'e', 'l', 'o'}
```