MAP, FILTER, ZIP

Higher order functions

A function that takes a function as a parameter and/or returns a function as its return value

Example: `sorted`

$\left.\begin{array}{l}\texttt{map}\\ \\ \texttt{filter}\end{array}\right\}$ modern alternative → list comprehensions and generator expressions

The map function

`map(func, *iterables)`

`*iterables` → a variable number of iterable objects

`func` → some function that takes as many arguments as there are iterable objects passed to `iterables`

`map(func, *iterables)` will then return an iterator that calculates the function applied to each element of the iterables

The iterator stops as soon as one of the iterables has been exhausted

so, unequal length iterables can be used

Examples

```
l = [2, 3, 4]

def sq(x):
    return x**2


list(map(sq, l))        → [4, 9, 16]




l1 = [1, 2, 3]
l2 = [10, 20, 30]

def add(x, y):
    return x + y

list(map(add, l1, l2))    → [11, 22, 33]


list(map(lambda x, y: x + y, l1, l2))   → [11, 22, 33]
```

The `filter` function

```
filter(func, iterable)
```

`iterable` → a single iterable

`func` → some function that takes a single argument

`filter(func, iterable)` will then return an iterator that contains all the elements of the iterable for which the function called on it is Truthy

If the function is `None`, it simply returns the elements of `iterable` that are Truthy

Examples

```
l = [0, 1, 2, 3, 4]

list(filter(None, l))     → [1, 2, 3, 4]

def is_even(n):
    return n % 2 == 0

list(filter(is_even, l))     → [0, 2, 4]

list(filter(lambda n: n % 2 == 0, l))     → [0, 2, 4]
```

The `zip` function

`zip(*iterables)`

`[1, 2, 3, 4]`

`[10, 20, 30, 40]`

$\xrightarrow{\texttt{zip}}$ `(1, 10), (2, 20), (3, 30), (4, 40)`

`[1, 2, 3]`

`[10, 20, 30]`

`['a', 'b', 'c']`

$\xrightarrow{\texttt{zip}}$ `(1, 10, 'a'), (2, 20, 'b'), (3, 30, 'c')`

`[1, 2, 3, 4, 5]`

`[10, 20, 30]`

$\xrightarrow{\texttt{zip}}$ `(1, 10), (2, 20), (3, 30)`

Examples

```
l1 = [1, 2, 3]
l2 = [10, 20, 30, 40]
l3 = 'python'

list(zip(l1, l2, l3))        → [(1, 10, 'p'), (2, 20, 'y'), (3, 30, 't')]
```

```
l1 = range(100)
l2 = 'abcd'

list(zip(l1, l2))            → [(0, 'a'), (1, 'b'), (2, 'c'), (3, 'd')]
```

List Comprehension Alternative to map

```
l = [2, 3, 4]

def sq(x):
    return x**2
                     }  list(map(lambda x: x**2, l))    → [4, 9, 16]
list(map(sq, l))
```

```
result = []
for x in l:
    result.append(x**2)    result → [4, 9, 16]
```

```
[x**2 for x in l]      → [4, 9, 16]

[<expression> for <varname> in <iterable>]
```

List Comprehension Alternative to map

```
l1 = [1, 2, 3]
l2 = [10, 20, 30]
```

```
list(map(lambda x, y: x + y, l1, l2))     → [11, 22, 33]
```

Remember: `zip(l1, l2)` → `[(1, 10), (2, 20), (3, 30)]`

```
[x + y for x, y in zip(l1, l2)]     → [11, 22, 33]
```

List Comprehension Alternative to filter

```
l = [1, 2, 3, 4]

list(filter(lambda n: n % 2 == 0, l))    → [2, 4]


[x for x in l if x % 2 == 0]    → [2, 4]


[<expression1> for <varname> in <iterable> if <expression2>]
```

Combining map and filter

```
l = range(10)
list(filter(lambda y: y < 25, map(lambda x: x**2, l)))      → [0, 1, 4, 9, 16]
```

Using a list comprehension is much clearer:

```
[x**2 for x in range(10) if x**2 < 25]     → [0, 1, 4, 9, 16]
```

Code