

Ch08-2-Lists-Advanced

September 10, 2025

1 List Comprehensions & Higher order functions

1.1 Topics

- list shortcuts
- lambda functions applications
- built-in higher order functions

1.2 List comprehension

- list is a very powerful and commonly used container
- list shortcuts can make you an efficient programmer
- E.g., an arithmetic set $S = \{x^2 : x \in \{0...9\}\}$
 - is equivalent to:
`S = [x**2 for x in range(10)]`
- consists of brackets containing an expression followed by a for clause, then zero or more for or if clauses
 - the expressions can be anything
 - always results a new list from evaluating expression
- syntax:

```
someList = [expression for item in list if conditional] # one-way selector
someList = [expression if conditionl else expression for item in list] # two-way selector
```

```
[1]: # Typical way to create a list of squared values of list 0 to 9?
sq = []
for i in range(10):
    sq.append(i**2)
```

```
[2]: print(sq)
```

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

```
[3]: # List comprehension -- handy technique:
S = [x**2 for x in range(10)]
```

```
[4]: S
```

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

In Math: $V = \{2^0, 2^1, 2^2, 2^3, \dots, 2^{12}\}$

[1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096]

In Math: $M = \{x|x \in S \text{ and } x \text{ even}\}$

```
[11]: # Simple approach in Python
M = []
for x in S:
    if x%2 == 0:
        M.append(x)
```

```
[12]: print(M)
```

[0, 4, 16, 36, 64]

```
[13]: # List comprehension
M1 = [x for x in S if x%2==0]
```

[14] : M1

```
[14]: [0, 4, 16, 36, 64]
```

```
[15]: assert M == M1, 'M and M1 are not equal!'
```

```
[17]: M2 = [True if x%2==0 else False for x in range(1, 21)]
```

[18] : M2

[illegible]

```
False,  
True]
```

```
[19]: sentence = "The quick brown fox jumps over the lazy dog"  
# words = sentence.split()  
# can make a list of tuples or list of lists  
wlist = [(w.upper(), w.lower(), len(w)) for w in sentence.split()]
```

```
[20]: wlist
```

```
[20]: [('THE', 'the', 3),  
      ('QUICK', 'quick', 5),  
      ('BROWN', 'brown', 5),  
      ('FOX', 'fox', 3),  
      ('JUMPS', 'jumps', 5),  
      ('OVER', 'over', 4),  
      ('THE', 'the', 3),  
      ('LAZY', 'lazy', 4),  
      ('DOG', 'dog', 3)]
```

1.3 Nested list comprehension

- syntax to handle the nested loop for nested lists

```
[21]: # let's create a nestedList of [[1, 2, 3, 4]*4]  
nestedList = [list(range(1, 5))]*5
```

```
[22]: nestedList
```

```
[22]: [[1, 2, 3, 4], [1, 2, 3, 4], [1, 2, 3, 4], [1, 2, 3, 4], [1, 2, 3, 4]]
```

```
[23]: # let's just keep the even values from each nested lists  
even = [x for lst in nestedList for x in lst if x%2==0 ]
```

```
[24]: even
```

```
[24]: [2, 4, 2, 4, 2, 4, 2, 4, 2, 4]
```

```
[25]: # let's create boolean single list of True/False  
evenOdd = [True if x%2 == 0 else False for lst in nestedList for x in lst]
```

```
[26]: evenOdd
```

```
[26]: [False,  
      True,  
      False,  
      True,  
      False,
```

```
True,  
False,  
True,  
False,  
True,  
False,  
True,  
False,  
True,  
False,  
True,  
False,  
True,  
False,  
True,  
False,  
True]
```

```
[29]: # let's create boolean nested list of True/False  
evenOdd1 = [[True if x%2 == 0 else False for x in lst] for lst in nestedList]
```

```
[30]: evenOdd1
```

```
[30]: [[False, True, False, True],  
       [False, True, False, True],  
       [False, True, False, True],  
       [False, True, False, True],  
       [False, True, False, True]]
```

1.4 higher order functions and lambda applications

- map, reduce, filter, sorted functions take function and iterable such as list as arguments
- lambda expression can be used as a parameter for higher order functions

1.4.1 sorted()

```
[31]: list1 = ['Apple', 'apple', 'ball', 'Ball', 'cat']  
list2 = sorted(list1)
```

```
[32]: print(list2)
```

```
['Apple', 'Ball', 'apple', 'ball', 'cat']
```

```
[33]: # sorting the list of tuples with different element (other than the first) as  
      ↪key  
list3 = [('cat', 10), ('ball', 20), ('apple', 3)]
```

```
[34]: # by default uses the first element as the key  
sorted(list3)
```

```
[34]: [('apple', 3), ('ball', 20), ('cat', 10)]
```

```
[35]: # check the original list
list3
```

```
[35]: [('cat', 10), ('ball', 20), ('apple', 3)]
```

```
[36]: # sorting the list of tuples with different element (other than the first) as
      ↪key
      # using itemgetter function
      from operator import itemgetter
      list5 = sorted(list3, key=itemgetter(1), reverse=True)
```

```
[37]: print(list5)
```

```
[('ball', 20), ('cat', 10), ('apple', 3)]
```

```
[38]: # directly using list item
      list6 = sorted(list3, key=lambda x: x[1], reverse=True)
```

```
[39]: print(list6)
```

```
[('ball', 20), ('cat', 10), ('apple', 3)]
```

1.4.2 filter()

- filter elements in the list by returning a new list for each element the function returns True

```
[40]: help(filter)
```

Help on class filter in module builtins:

```
class filter(object)
|   filter(function or None, iterable) --> filter object
|
|   Return an iterator yielding those items of iterable for which function(item)
|   is true. If function is None, return the items that are true.
|
|   Methods defined here:
|
|   __getattr__(self, name, /)
|       Return getattr(self, name).
|
|   __iter__(self, /)
|       Implement iter(self).
|
|   __next__(self, /)
|       Implement next(self).
|
|   __reduce__(...)
```

```

|         Return state information for pickling.
|
| -----
| Static methods defined here:
|
| __new__(*args, **kwargs) from builtins.type
|     Create and return a new object.  See help(type) for accurate signature.

```

```
[41]: list7 = [2, 18, 9, 22, 17, 24, 8, 12, 27]
      list8 = list(filter(lambda x: x%3==0, list7))
```

```
[42]: print(list8)
```

```
[18, 9, 24, 12, 27]
```

1.4.3 map()

```
[43]: help(map)
```

Help on class map in module builtins:

```

class map(object)
|   map(func, *iterables) --> map object
|
|   Make an iterator that computes the function using arguments from
|   each of the iterables.  Stops when the shortest iterable is exhausted.
|
|   Methods defined here:
|
|   __getattr__(self, name, /)
|       Return getattr(self, name).
|
|   __iter__(self, /)
|       Implement iter(self).
|
|   __next__(self, /)
|       Implement next(self).
|
|   __reduce__(...)
|       Return state information for pickling.
|
| -----
| Static methods defined here:
|
| __new__(*args, **kwargs) from builtins.type
|     Create and return a new object.  See help(type) for accurate signature.

```

```
[44]: items = list(range(1, 11))
      squared = list(map(lambda x: x**2, items))
```

```
[45]: print(squared)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

```
[46]: # map each words with its length
      sentence = "The quick brown fox jumps over the lazy dog"
      words = [word.lower() for word in sentence.split()]
```

```
[47]: print(words)
```

```
['the', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog']
```

```
[48]: w_len = list(map(lambda w: (w, w.upper(), len(w)), words))
```

```
[49]: print(w_len)
```

```
[('the', 'THE', 3), ('quick', 'QUICK', 5), ('brown', 'BROWN', 5), ('fox', 'FOX', 3), ('jumps', 'JUMPS', 5), ('over', 'OVER', 4), ('the', 'THE', 3), ('lazy', 'LAZY', 4), ('dog', 'DOG', 3)]
```

1.4.4 mapping one type to another

```
[50]: # Example: map string to integers; common operation while reading list of
      ↪ numbers
      data = input('Enter numbers separated by space: ')
```

```
Enter numbers separated by space: 1 2 100 99 50
```

```
[51]: data
```

```
[51]: '1 2 100 99 50'
```

```
[54]: nums = list(map(int, input().split()))
```

```
100 99 45 454 454 4545 455
```

```
[55]: nums
```

```
[55]: [100, 99, 45, 454, 454, 4545, 455]
```

1.4.5 reduce()

- used to reduce a list of values to a single output
- reduce() is defined in functools module

```
[56]: import functools
      help(functools)
```

Help on module functools:

NAME

functools - functools.py - Tools for working with functions and callable objects

MODULE REFERENCE

<https://docs.python.org/3.9/library/functools>

The following documentation is automatically generated from the Python source files. It may be incomplete, incorrect or include features that are considered implementation detail and may vary between Python implementations. When in doubt, consult the module reference at the location listed above.

CLASSES

```
builtins.object
    cached_property
    partial
    partialmethod
    singledispatchmethod
```

```
class cached_property(builtins.object)
|   cached_property(func)
|
|   Methods defined here:
|
|   __get__(self, instance, owner=None)
|
|   __init__(self, func)
|       Initialize self. See help(type(self)) for accurate signature.
|
|   __set_name__(self, owner, name)
|
|   -----
|   Class methods defined here:
|
|   __class_getitem__ = GenericAlias(...) from builtins.type
|       Represent a PEP 585 generic type
|
|       E.g. for t = list[int], t.__origin__ is list and t.__args__ is
(int,).
|
|   -----
|   Data descriptors defined here:
|
|   __dict__
|       dictionary for instance variables (if defined)
```



```

|
|  __weakref__
|      list of weak references to the object (if defined)
|
class partial(builtins.object)
|  partial(func, *args, **keywords) - new function with partial application
|  of the given arguments and keywords.
|
|  Methods defined here:
|
|  __call__(self, /, *args, **kwargs)
|      Call self as a function.
|
|  __delattr__(self, name, /)
|      Implement delattr(self, name).
|
|  __getattr__(self, name, /)
|      Return getattr(self, name).
|
|  __reduce__(...)
|      Helper for pickle.
|
|  __repr__(self, /)
|      Return repr(self).
|
|  __setattr__(self, name, value, /)
|      Implement setattr(self, name, value).
|
|  __setstate__(...)
|
|  -----
|  Class methods defined here:
|
|  __class_getitem__(...) from builtins.type
|      See PEP 585
|
|  -----
|  Static methods defined here:
|
|  __new__(*args, **kwargs) from builtins.type
|      Create and return a new object.  See help(type) for accurate
signature.
|
|  -----
|  Data descriptors defined here:
|
|  __dict__
|

```

```

|   args
|       tuple of arguments to future partial calls
|
|   func
|       function object to use in future partial calls
|
|   keywords
|       dictionary of keyword arguments to future partial calls
|
class partialmethod(builtins.object)
|   partialmethod(func, /, *args, **keywords)
|
|   Method descriptor with partial application of the given arguments
|   and keywords.
|
|   Supports wrapping existing descriptors and handles non-descriptor
|   callables as instance methods.
|
|   Methods defined here:
|
|   __get__(self, obj, cls=None)
|
|   __init__(self, func, /, *args, **keywords)
|       Initialize self.  See help(type(self)) for accurate signature.
|
|   __repr__(self)
|       Return repr(self).
|
|   -----
|   Class methods defined here:
|
|   __class_getitem__ = GenericAlias(...) from builtins.type
|       Represent a PEP 585 generic type
|
|       E.g. for t = list[int], t.__origin__ is list and t.__args__ is
(int,).
|
|   -----
|   Readonly properties defined here:
|
|   __isabstractmethod__
|
|   -----
|   Data descriptors defined here:
|
|   __dict__
|       dictionary for instance variables (if defined)
|

```

```

|   __weakref__
|       list of weak references to the object (if defined)

class singledispatchmethod(builtins.object)
|   singledispatchmethod(func)
|
|   Single-dispatch generic method descriptor.
|
|   Supports wrapping existing descriptors and handles non-descriptor
|   callables as instance methods.
|
|   Methods defined here:
|
|   __get__(self, obj, cls=None)
|
|   __init__(self, func)
|       Initialize self.  See help(type(self)) for accurate signature.
|
|   register(self, cls, method=None)
|       generic_method.register(cls, func) -> func
|
|       Registers a new implementation for the given *cls* on a
*generic_method*.
|
|   -----
|   Readonly properties defined here:
|
|   __isabstractmethod__
|
|   -----
|   Data descriptors defined here:
|
|   __dict__
|       dictionary for instance variables (if defined)
|
|   __weakref__
|       list of weak references to the object (if defined)

```

FUNCTIONS

```

cache(user_function, /)
    Simple lightweight unbounded cache.  Sometimes called "memoize".

cmp_to_key(...)
    Convert a cmp= function into a key= function.

lru_cache(maxsize=128, typed=False)
    Least-recently-used cache decorator.

```

If `*maxsize*` is set to `None`, the LRU features are disabled and the cache can grow without bound.

If `*typed*` is `True`, arguments of different types will be cached separately.

For example, `f(3.0)` and `f(3)` will be treated as distinct calls with distinct results.

Arguments to the cached function must be hashable.

View the cache statistics named tuple (`hits`, `misses`, `maxsize`, `currsize`) with `f.cache_info()`. Clear the cache and statistics with `f.cache_clear()`.

Access the underlying function with `f.__wrapped__`.

See: [https://en.wikipedia.org/wiki/Cache_replacement_policies#Least_recently_used_\(LRU\)](https://en.wikipedia.org/wiki/Cache_replacement_policies#Least_recently_used_(LRU))

`reduce(...)`
`reduce(function, sequence[, initial]) -> value`

Apply a function of two arguments cumulatively to the items of a sequence, from left to right, so as to reduce the sequence to a single value. For example, `reduce(lambda x, y: x+y, [1, 2, 3, 4, 5])` calculates `((((1+2)+3)+4)+5)`. If `initial` is present, it is placed before the items of the sequence in the calculation, and serves as a default when the sequence is empty.

`singledispatch(func)`
Single-dispatch generic function decorator.

Transforms a function into a generic function, which can have different behaviours depending upon the type of its first argument. The decorated function acts as the default implementation, and additional implementations can be registered using the `register()` attribute of the generic function.

`total_ordering(cls)`
Class decorator that fills in missing ordering methods

`update_wrapper(wrapper, wrapped, assigned=('__module__', '__name__', '__qualname__', '__doc__', '__annotations__'), updated=('__dict__',))`
Update a wrapper function to look like the wrapped function

`wrapper` is the function to be updated
`wrapped` is the original function
`assigned` is a tuple naming the attributes assigned directly

from the wrapped function to the wrapper function (defaults to `functools.WRAPPER_ASSIGNMENTS`)
updated is a tuple naming the attributes of the wrapper that are updated with the corresponding attribute from the wrapped function (defaults to `functools.WRAPPER_UPDATES`)

```
wraps(wrapped, assigned=('__module__', '__name__', '__qualname__',  
 '__doc__', '__annotations__'), updated=('__dict__',))  
Decorator factory to apply update_wrapper() to a wrapper function
```

Returns a decorator that invokes `update_wrapper()` with the decorated function as the wrapper argument and the arguments to `wraps()` as the remaining arguments. Default arguments are as for `update_wrapper()`. This is a convenience function to simplify applying `partial()` to `update_wrapper()`.

DATA

```
WRAPPER_ASSIGNMENTS = ('__module__', '__name__', '__qualname__', '__do...  
WRAPPER_UPDATES = ('__dict__',)  
__all__ = ['update_wrapper', 'wraps', 'WRAPPER_ASSIGNMENTS', 'WRAPPER_...
```

FILE

```
/Users/rbasnet/miniconda3/envs/py/lib/python3.9/functools.py
```

1.4.6 reduce applications

1.4.7 find sum of first n positive integers

```
[57]: s = functools.reduce(lambda x,y:x+y, range(1, 11))
```

```
[58]: # test the result!  
assert sum(range(1, 11)) == s
```

1.4.8 find factorial (or product) of first n positive integers

```
[59]: fact = functools.reduce(lambda x,y:x*y, range(1, 11))
```

```
[60]: fact
```

```
[60]: 3628800
```

```
[61]: # test the result using math.factorial function  
import math  
assert math.factorial(10) == fact
```

1.5 Kattis problems

1. Connect-N - <https://open.kattis.com/problems/connectn>
 - Hint: 2-D Array - simply check 4 winning ways from each B or R char - just like in tic-tac-toe

[]: