In [1]:
```
# Lambda expression to pass a function to a function
# Examples with filter, map and sorted
```

In [2]:
```
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:
 |
 |  __getattribute__(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.
```

In [3]:
```
# Example - create a tuple from the list where marks is greater than 80.

marks = [55, 87, 99, 63, 89, 48, 85, 76]

highMarks = tuple(filter(lambda x: x > 80, marks))

print(highMarks)
```

```
(87, 99, 89, 85)
```

In [4]:
```
# Example - create a sorted list of marks greater than 80.

marks = [55, 87, 99, 63, 89, 48, 85, 76]

sortedHighMarks = sorted(filter(lambda x: x > 80, marks))

print(sortedHighMarks)
```

```
[85, 87, 89, 99]
```

In [5]:
```
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:
 |
 |  __getattribute__(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.
```

In [6]:
```
# Example - create a tuple from the given list to
# scale the marks out of 40 instead of 100

marks = [55, 77, 99, 64]

highMarks = tuple(map(lambda x: x * .4, marks))

print(highMarks)
```

```
(22.0, 30.8, 39.6, 25.6)
```

In [7]:
```
help(sorted)
```

```
Help on built-in function sorted in module builtins:

sorted(iterable, /, *, key=None, reverse=False)
    Return a new list containing all items from the iterable in ascending order.

    A custom key function can be supplied to customize the sort order, and the
    reverse flag can be set to request the result in descending order.
```

In [8]:
```
cities = ['Kolkata','Bardhaman','Malda','Jalpaiguri']
sortedCities = sorted(cities)
print(sortedCities)
```

```
['Bardhaman', 'Jalpaiguri', 'Kolkata', 'Malda']
```

In [9]:
```
# Sorting the cities by length of the city name

cities = ['Kolkata','Bardhaman','Malda','Jalpaiguri']

sortedCities = sorted(cities, key = len)

print(sortedCities)
```

```
['Malda', 'Kolkata', 'Bardhaman', 'Jalpaiguri']
```

In [10]:
```
# sorting a list of tuples

marks = [('Sayani',40,60,80), ('Aditi',30,70,90),('Amirul',80,65,55)]

sorted_marks = sorted(marks)

print(sorted_marks)
```

```
[('Aditi', 30, 70, 90), ('Amirul', 80, 65, 55), ('Sayani', 40, 60, 80)]
```

In [11]:
```
# sorting by the third marks in each tuple

marks = [('Sayani',40,60,80), ('Aditi',30, 70, 90),('Amirul',80, 65, 55)]

sorted_marks = sorted(marks, key = lambda x : x[3])

print(sorted_marks)
```

```
[('Amirul', 80, 65, 55), ('Sayani', 40, 60, 80), ('Aditi', 30, 70, 90)]
```

In [12]:
```
# descending sort by the third marks in each tuple

marks = [('Sayani',40,60,80), ('Aditi',30, 70, 90),('Amirul',80, 65, 55)]

sorted_marks = sorted(marks, key = lambda x : x[3], reverse=True)

print(sorted_marks)
```

```
[('Aditi', 30, 70, 90), ('Sayani', 40, 60, 80), ('Amirul', 80, 65, 55)]
```

In [13]:
```
# sorting by the second, third marks in each tuple

marks = [('Sayani',40,60,80), ('Aditi',30, 50, 90),('Amirul',80, 60, 55)]

sorted_marks = sorted(marks, key = lambda x : (x[2],x[3]))

print(sorted_marks)
```

```
[('Aditi', 30, 50, 90), ('Amirul', 80, 60, 55), ('Sayani', 40, 60, 80)]
```

In [14]:
```
# sorting by the sum of the marks

marks = [('Sayani',[40,60,80]), ('Aditi',[30, 50, 90]),('Amirul',[80, 60, 55])]

sorted_marks = sorted(marks, key = lambda x : sum(x[1]))

print(sorted_marks)
```

```
[('Aditi', [30, 50, 90]), ('Sayani', [40, 60, 80]), ('Amirul', [80, 60, 55])]
```

In [15]:
```
# sorting the items in a dictionary by key

marks = {"Aditi":73, "Zahir":42, "Anand":59}
sorted_marks = dict(sorted(marks.items()))
print(sorted_marks)
```

```
{'Aditi': 73, 'Anand': 59, 'Zahir': 42}
```

In [16]:
```
# sorting the items in a dictionary by value

marks = {"Aditi":73, "Zahir":42, "Anand":59}

sorted_marks = dict(sorted(marks.items(), key = lambda x : x[1]))
```

```
print(sorted_marks)
```

```
{'Zahir': 42, 'Anand': 59, 'Aditi': 73}
```

In [ ]:

```
'''
all variable 'assignments' in a function store the value in
the local symbol table;

whereas variable 'references'
first look in the local symbol table, then
in the local symbol tables of enclosing functions,
then in the global symbol table,
and finally in the table of built-in names.

Thus, global variables and variables of enclosing functions
cannot be directly assigned a value within a function
(unless, for global variables, using 'global; or,
for variables of enclosing functions, using 'nonlocal'),
although they may be referenced.
'''
```

In [17]:

```
# an assignment statement in a function creates a local variable
# this variable only exists inside the function and cannot be used outside it
# when the execution of the function terminates (returns),
# the local variables are destroyed.
# formal parameters are also local

def f(x):
    num1 = 5
    return x - num1

print(f(13))
print(num1)
```

```
8
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[17], line 12
      9     return x - num1
     11 print(f(13))
---> 12 print(num1)

NameError: name 'num1' is not defined
```

In [18]:

```
# it is legal for a function to 'access' a global variable

def f(x):
    print(y)
    return x - y

y = 5
print(f(13))
```

```
5
8
```

In [19]:

```
# variable 'references' in a function first look in the local symbol table.
# A new local variable is created in the function’s namespace if another
# value is assigned to a globally declared variable inside the function.
# The value of the global variable will not be changed.
# So, when a local variable has the same name as a global variable,
# the local variable is accessed when the variable name
# in the function is referenced
```

```
y = 5            # this y is a global variable

def f(x):
    y = 10       # this y is a local variable
    z = x - y    # local y is accessed i.e 10, not 5
    return z


print(f(13))
print(y)
```

```
3
5
```

In [20]:
```
y=5                 # this y is a global variable

def f(x):
    y = y - 1       # local variable 'y' is referenced in the local scope
                    # before it is assigned a value
    return x - y

print(f(13))
```

```
---------------------------------------------------------------------------
UnboundLocalError                         Traceback (most recent call last)
Cell In[20], line 8
      5                     # before it is assigned a value
      6     return x - y
----> 8 print(f(13))

Cell In[20], line 4, in f(x)
      3 def f(x):
----> 4     y = y - 1       # local variable 'y' is referenced in the local scope
      5                     # before it is assigned a value
      6     return x - y

UnboundLocalError: local variable 'y' referenced before assignment
```

In [22]:
```
y=5                 # this y is a global variable

def f(x):
    global y
    y = 10
    return x - y

print(f(13))
print(y)
```

```
3
10
```