Lambda

A `lambda` operator or `lambda` function is used for creating small, one-time, anonymous function objects in Python.

Basic Syntax

```
lambda arguments : expression
```

A `lambda` operator can have any number of arguments but can have only one expression. It cannot contain any statements and returns a function object which can be assigned to any variable.

Example

Let’s look at a function in Python:

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

# Call the function
add(2, 3)  # Output: 5
```

The above function’s name is `add`, it expects two arguments `x` and `y` and returns their sum.

Let’s see how we can convert the above function into a `lambda` function:

```
add = lambda x, y : x + y

print add(2, 3) # Output: 5
```

In `lambda x, y: x + y;` `x` and `y` are arguments to the function and `x + y` is the expression that gets executed and its values are returned as output.

`lambda x, y: x + y` returns a function object which can be assigned to any variable, in this case, the function object is assigned to the `add` variable.

```
type (add) # Output: function
```

Map

Basic Syntax

```
map(function_object, iterable1, iterable2,...)
```

`map` functions expect a function object and any number of `iterables`, such as `list`, `dictionary`, etc. It executes the `function_object` for each element in the sequence and returns a `list` of the elements modified by the function object.

```
def multiply2(x):
  return x * 2

map(multiply2, [1, 2, 3, 4])  # Output [2, 4, 6, 8]
```

In the above example, `map` executes the `multiply2` function for each element in the list, `[1, 2, 3, 4]`, and returns `[2, 4, 6, 8]`.

Let’s see how we can write the above code using `map` and `lambda`.

```
map(lambda x : x*2, [1, 2, 3, 4]) #Output [2, 4, 6, 8]
```

Iterating Over a Dictionary Using Map and Lambda

```
dict_a = [{'name': 'python', 'points': 10}, {'name': 'java', 'points': 8}]

map(lambda x : x['name'], dict_a) # Output: ['python', 'java']

map(lambda x : x['points']*10,  dict_a) # Output: [100, 80]

map(lambda x : x['name'] == "python", dict_a) # Output: [True, False]
```

In the above example, each `dict` of `dict_a` will be passed as a parameter to the `lambda` function. The result of the `lambda` function expression for each `dict` will be given as output.

Multiple Iterables to the Map Function

We can pass multiple sequences to the `map` functions as shown below:

```
list_a = [1, 2, 3]
list_b = [10, 20, 30]

map(lambda x, y: x + y, list_a, list_b) # Output: [11, 22, 33]
```

Here, each `i^th` element of `list_a` and `list_b` will be passed as an argument to the `lambda` function.

We can't access the elements of the `map object` with index nor we can use `len()` to find the length of the `map object`.

We can, however, force convert the `map` output, i.e. the `map object`, to `list` as shown below:

```
map_output = map(lambda x: x*2, [1, 2, 3, 4])
print(map_output) # Output: map object: <map object at 0x04D6BAB0>

list_map_output = list(map_output)

print(list_map_output) # Output: [2, 4, 6, 8]
```

Filter

Basic Syntax

```
filter(function_object, iterable)
```

The `filter` function expects two arguments: `function_object` and an iterable. `function_object` returns a boolean value and is called for each element of the `iterable`. `filter` returns only those elements for which the `function_object` returns `True`.

Like the `map` function, the `filter` function also returns a list of elements. Unlike `map`, *the* `filter` function can only have one `iterable` as input.

Even number using `filter` function:

```
a = [1, 2, 3, 4, 5, 6]
filter(lambda x : x % 2 == 0, a) # Output: [2, 4, 6]
```

Filter `list` of `dicts`:

```
dict_a = [{'name': 'python', 'points': 10}, {'name': 'java', 'points': 8}]

filter(lambda x : x['name'] == 'python', dict_a) # Output: [{'name': 'python', 'points': 10}]
```

Similar to `map`, the `filter` function in Python3 returns a `filter object` or the iterator which gets lazily evaluated. We cannot access the elements of the `filter object` with index, nor can we use `len()` to find the length of the `filter object`.

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

filter_obj = filter(lambda x: x % 2 == 0, list_a)
even_num = list(filter_obj) # Converts the filer obj to a list

print(even_num) # Output: [2, 4]
```