Class 7

“lambda” function

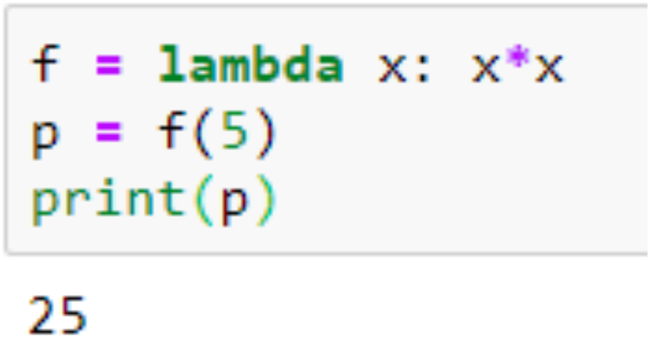
Functions: “lambda” expressions

- **lambda** expressions are another way of defining functions but with a difference.
  - They aren't capable of multiple expressions and can only handle single expressions.
  - They can only be used one time.
- Syntax:
  - lambda arguments: expression → lambda function
- Comparing with function structure
  - def functionName(arguments): → Regular function

```
def functionName( arguments ):
        statements...
        return something
```

“lambda” function examples:

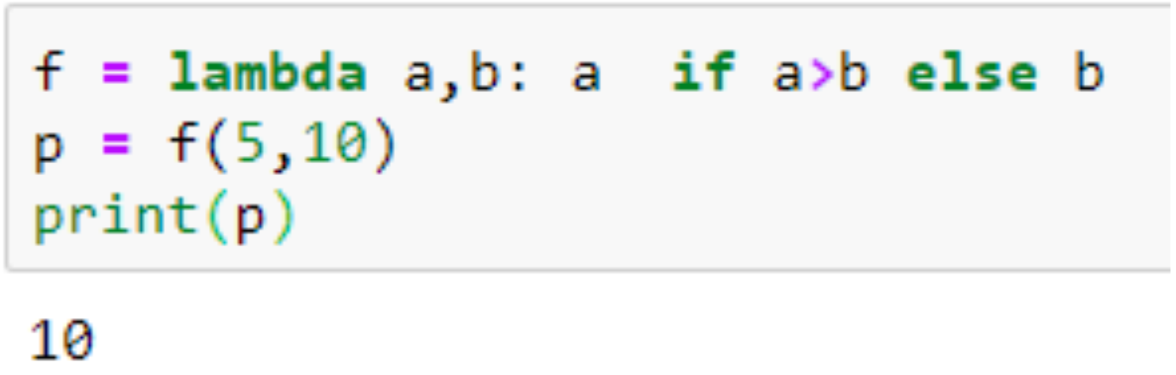
- Compute the square of a number using lambda function



```python
f = lambda x: x*x
p = f(5)
print(p)
```

```
25
```

- Find out which number is greater of 2 using lambda function



```python
f = lambda a,b: a  if a>b else b
p = f(5,10)
print(p)
```

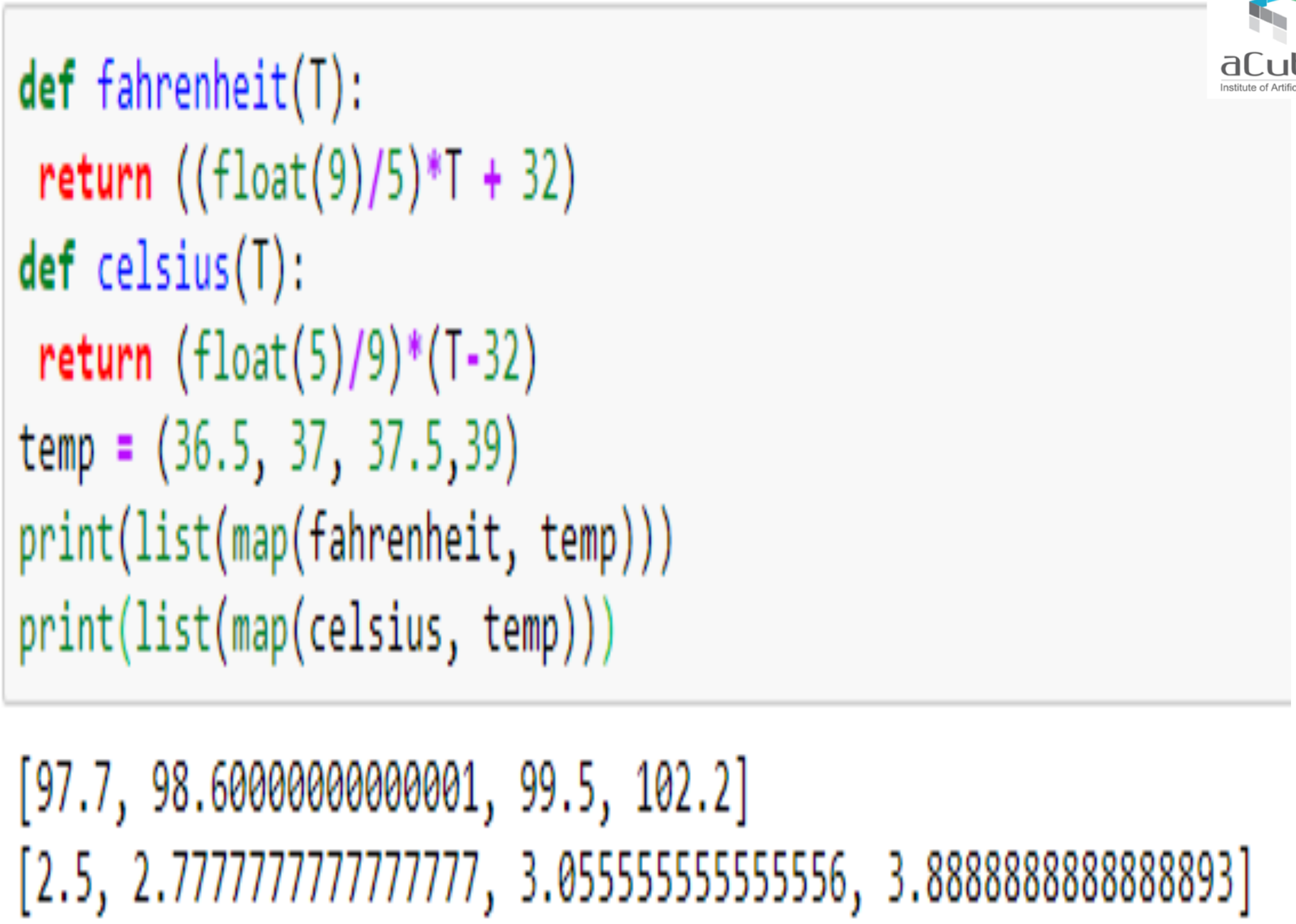
```
10
```

Map, filter, reduce functions

‘map’ function

- **The map() function:** map() is a function with two arguments.
  - Syntax: r = map(func, seq)
- The advantage of the lambda operator can be seen when it is used in combination with the map() function.

- The first argument func is the name of a function and the second a sequence (e.g. a list) seq.
- map() applies the function func to all the elements of the sequence seq.
- It returns a new list with the elements changed by function



```python
def fahrenheit(T):
 return ((float(9)/5)*T + 32)
def celsius(T):
 return (float(5)/9)*(T-32)
temp = (36.5, 37, 37.5,39)
print(list(map(fahrenheit, temp)))
print(list(map(celsius, temp)))
```

```
[97.7, 98.60000000000001, 99.5, 102.2]
[2.5, 2.7777777777777777, 3.0555555555555556, 3.8888888888888893]
```

- map() can be applied to **more than one list**.
- The lists have to **have the same length**.
- map() will apply its lambda function to the respective elements of the argument lists
- It first applies to the elements with the 0th index, then to the elements with the 1st index until the nth index is reached:

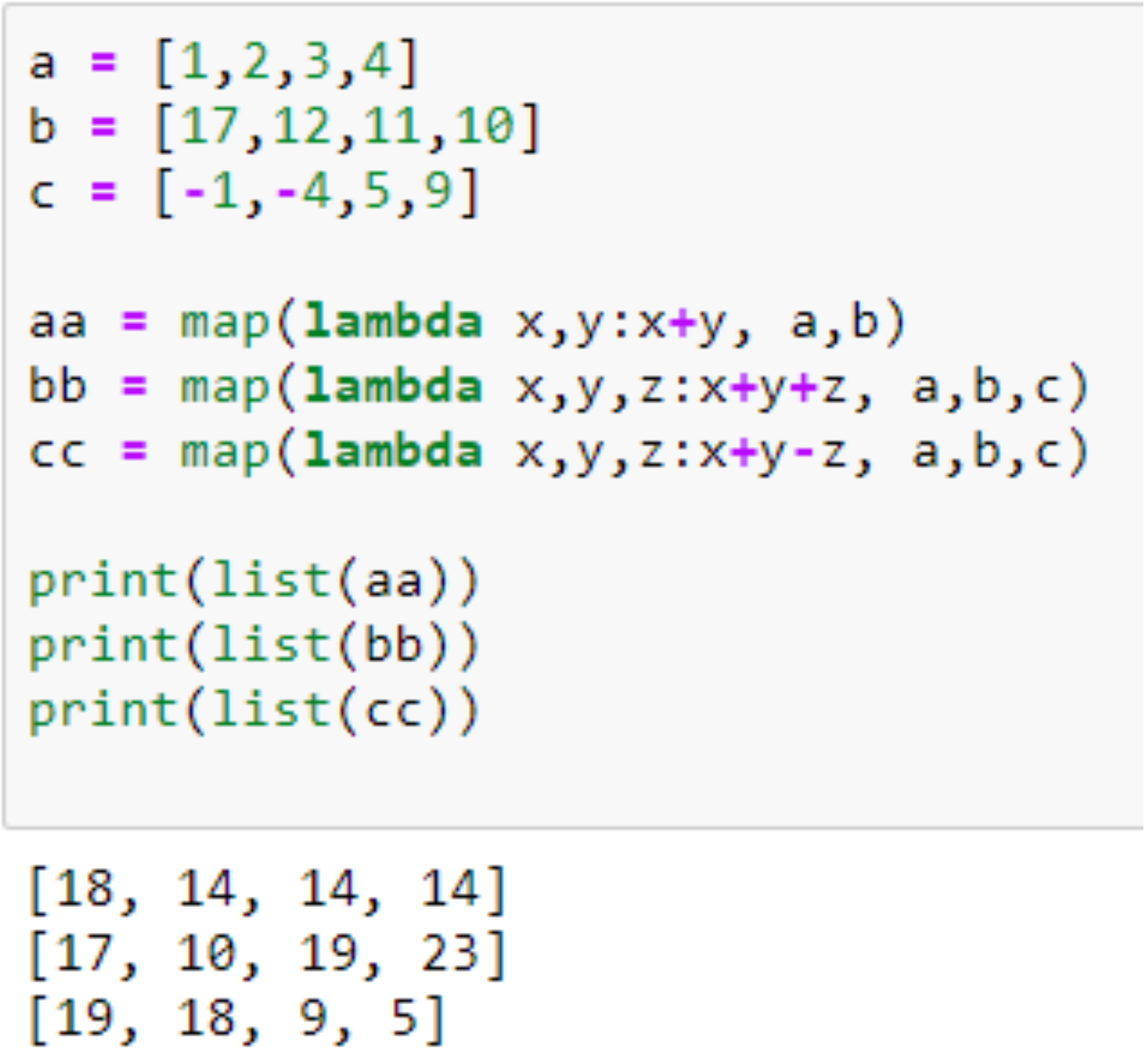
Exercise on ‘map’:

1. a = [1,2,3,4] b = [17,12,11,10] c = [-1,-4,5,9]
   - 1. Compute element-wise sum of a and b.
   - 2. Compute element-wise sum of a, b, and c.
   - 3. Compute a + b – c (element-wise).

We can see in the example above that the parameter x gets its values from the list a, while y gets its values from b, and z from list c

- **Apply lambda function**

aCubeIT
Institute of Artificial Intelligence



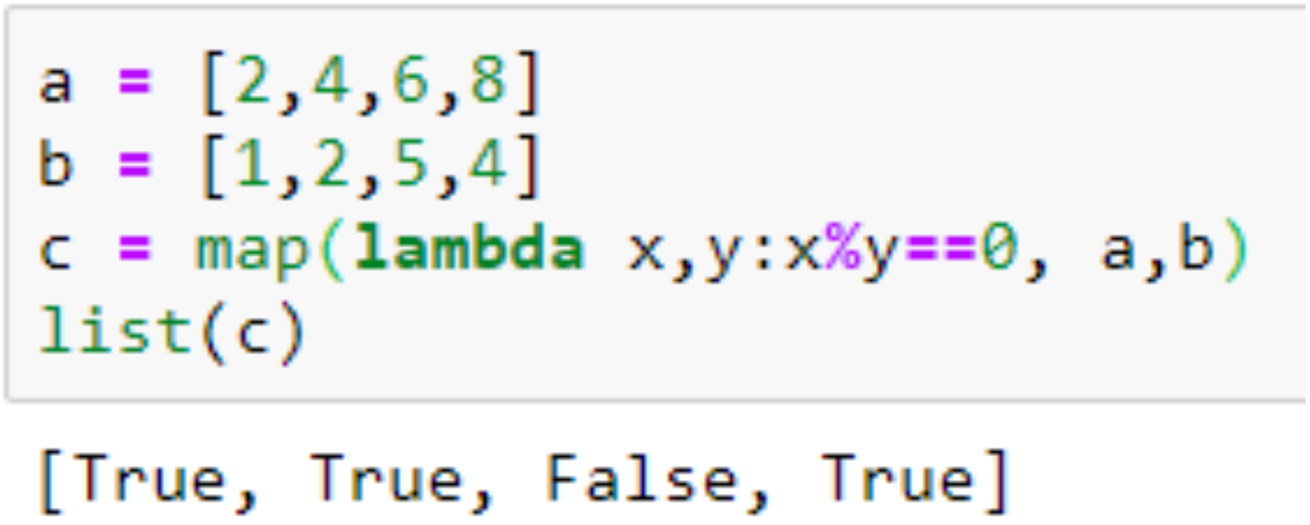
```
a = [1,2,3,4]
b = [17,12,11,10]
c = [-1,-4,5,9]

aa = map(lambda x,y:x+y, a,b)
bb = map(lambda x,y,z:x+y+z, a,b,c)
cc = map(lambda x,y,z:x+y-z, a,b,c)

print(list(aa))
print(list(bb))
print(list(cc))
```

```
[18, 14, 14, 14]
[17, 10, 19, 23]
[19, 18, 9, 5]
```

2. Check whether each element in ‘b’ is a factor of respective element in ‘a’

- a = [2,4,6,8]
- b = [1,2,5,4]

- Use **lambda function**



```python
a = [2,4,6,8]
b = [1,2,5,4]
c = map(lambda x,y:x%y==0, a,b)
list(c)
```

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

‘filter’ function

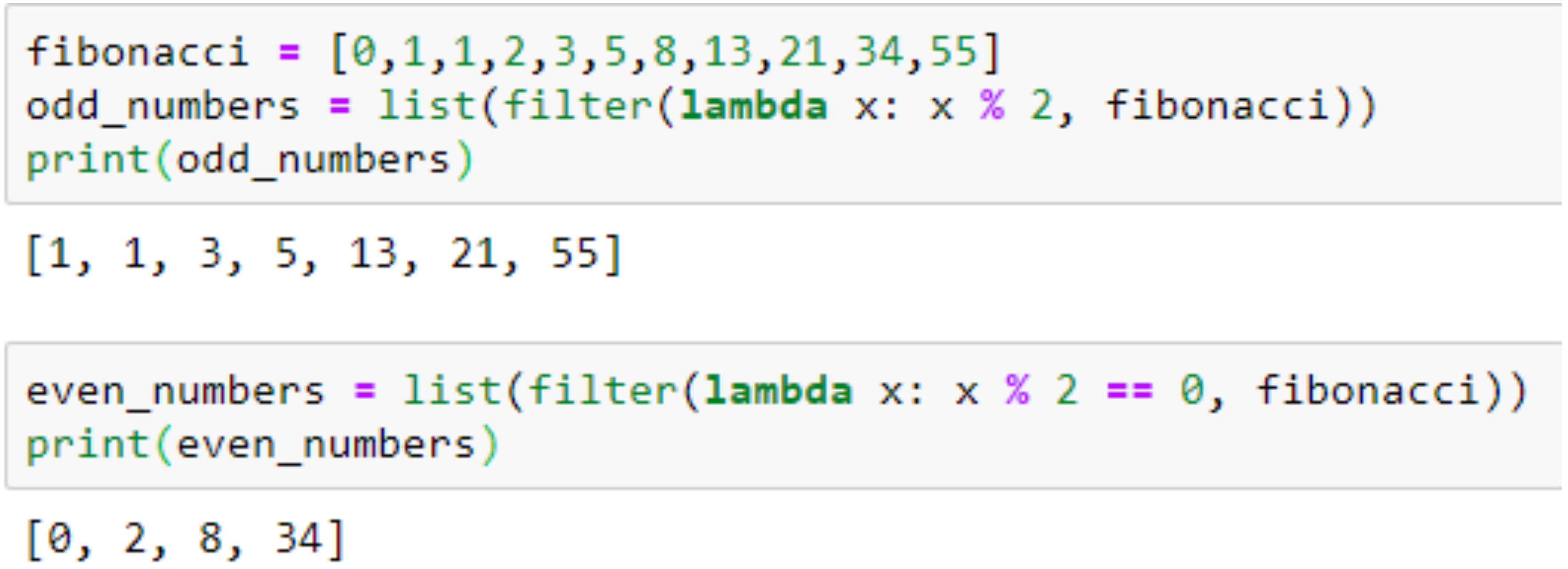
- The function **filter(function, list)**
- It offers an elegant way to filter out all the elements of a list, for which the function returns **True**.
- The function **filter(f,l)** needs a function **‘f’** as its first argument.
- **‘f’** returns a Boolean value, i.e. either **True** or **False**.
- This function will be applied to every element of the list ***‘l’***.
- Only if **‘f’** returns True will the element of the list be included in the result list

‘filter’ Example:

1. fib = [0,1,1,2,3,5,8,13,21,34,55]

A. Return odd Fibonacci numbers from the following sequence of Fibonacci numbers

B. Return even Fibonacci numbers from the above sequence of Fibonacci numbers:



```
fibonacci = [0,1,1,2,3,5,8,13,21,34,55]
odd_numbers = list(filter(lambda x: x % 2, fibonacci))
print(odd_numbers)
```

```
[1, 1, 3, 5, 13, 21, 55]
```

```
even_numbers = list(filter(lambda x: x % 2 == 0, fibonacci))
print(even_numbers)
```

```
[0, 2, 8, 34]
```

2. a = [1,3,6,9,18,36,24]

a) Get the list of values divisible by both 3 and 6

b) Get the list of values divisible by both 3 or 9

c) Get the list of values divisible by 2 and 3

3. b = ['44', '32', '55', '23', '51', '66‘, ’a2a’]

d) Get the list of strings where first and last letters are same

e) Get the list of strings where first and last letters are not same

‘reduce’ function

- The function **reduce(func, seq)** continually applies the function,’func()’ to the sequence ‘seq’.

- It returns a single value.

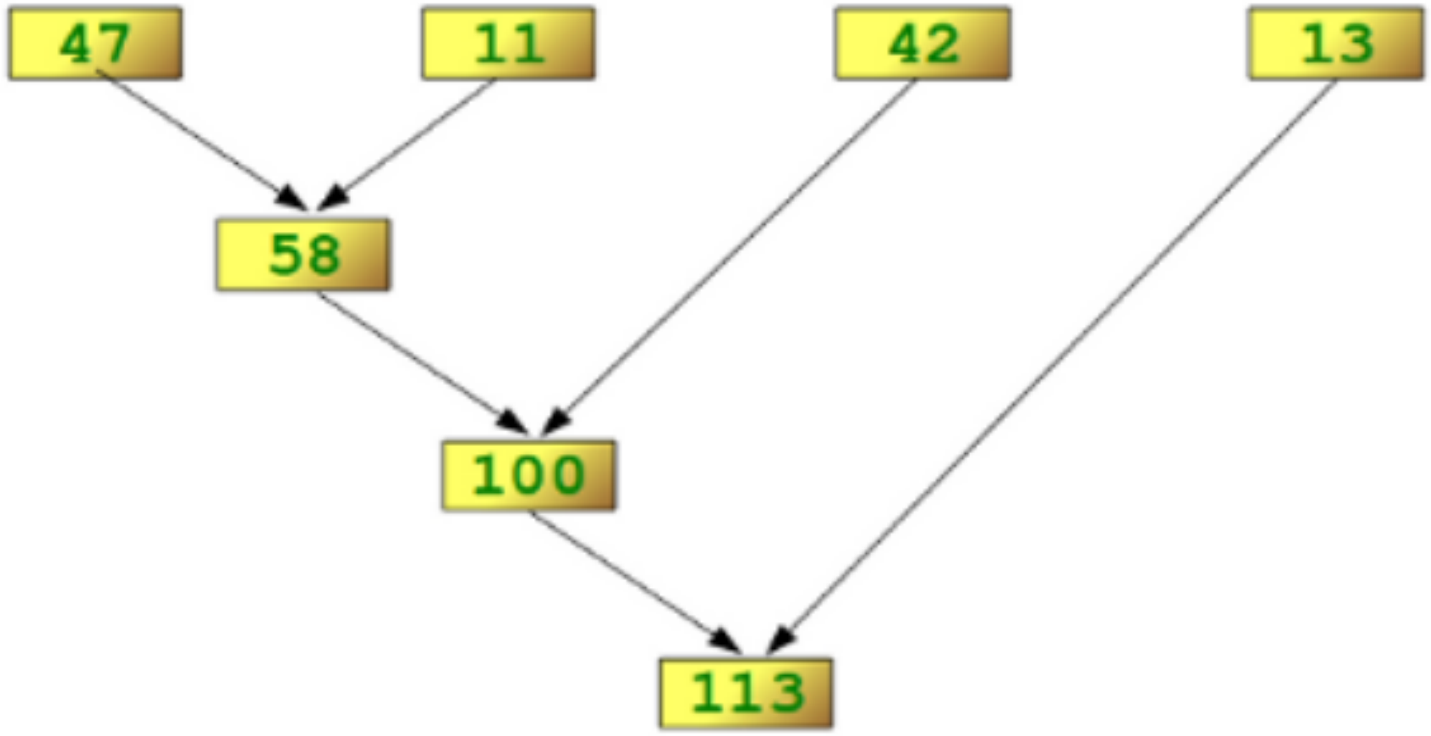
If seq = [s1, s2, s3, ... , sn], calling reduce(func, seq) works like this:

1. The first two elements of seq will be applied to func, i.e. **func(s1,s2)**
2. The list on which reduce() works looks now like this: [**func(**s1, s2**), s3, ... , sn]**
3. In the next step, func will be applied on the previous result and the third element of the list, i.e. **func(**func(s1, s2)**,s3)**
4. The list looks like this now: [**func**(func(s1, s2),s3), ... **, sn]**
5. Continue like this until just one element is less and return this element as the result of reduce()

```python
from functools import reduce
reduce(lambda x,y: x+y, [47,11,42,13])
```

```
113
```

The following diagram shows the intermediate steps of the calculation:


$$\begin{tikzcd}
47 \arrow[dr] & & 11 \arrow[dl] & 42 \arrow[ddl] & 13 \arrow[dddl] \\
& 58 \arrow[dr] & & & \\
& & 100 \arrow[dr] & & \\
& & & 113 &
\end{tikzcd}$$

‘reduce’ Example:

1. Determining the maximum of a list of numerical values by using reduce. **(from functools import reduce)**
2. Calculating the sum of the numbers from 1 to 100
3. Calculate the multiple of numbers from 1 to 10.