# lambda expression or anonymous function

* A simple 1 line function
* doesn’t use return or def keywords, they are implicit.

names = ['John Doe', 'Noolmyan Rathoz', 'Conor Lazi', 'Micheal Johnson']

names.sort(key = lambda name: name.split(' ')[-1].lower())

print(names)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# def add(x, y):

# return x + y

add = lambda x, y: x + y

print(add(10, 12))

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

def mx(x, y):

if x > y:

return x

else:

return y

mx = lambda x, y: x if x > y else y

res = mx(1, 10)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Why Use Lambda Functions?

The power of lambda is better shown when you use them as an anonymous function inside another function.

Say you have a function definition that takes one argument, and that argument will be multiplied with an unknown number:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

def myfunc(n):

return lambda a: a \* n

doubler = myfunc(2) # doubler = lambda a: a \* 2

doubler\_res = doubler(3) # doubler\_res = 3 \* 2

print(doubler\_res) # 6

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

def myfunc(n):  
  return lambda a : a \* n  
  
mydoubler = myfunc(2)  
mytripler = myfunc(3)  
  
print(mydoubler(11)) # 22   
print(mytripler(11)) # 33

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_