Anonymous Functions (Not identified by name or Unknown name)
It has the following alias Names:
1 Lambda functions
2 Lambda Forms
3 One Line function
4 Implict functions
5 Simple functions
6 Throw-away functions
7 Short period functions
.................................!!
It is a function that is defined without a name. No def and return keywords..!!

Syntax
```
lambda [arg1 [,arg2,.....argn]]:expression
```

Example:
```
n=lambda x:2*x
n ==> It is a variable or identifier
lambda ==> It is a keyword
x ==> Argument
: ==> Delimiter
2*x ==> It is an Expression
```

Example:
```
def add(x,y):
    return(x+y)
add(1,2)
```

Example:
```
add=lambda x,y:x+y
add(4,3)
```

Example:
```
print((lambda x,y:x+y)(2,3))
```

Example
```
g=lambda x:x*x
print(g(5))
```

Example:
```
def average(x, y):
    return (x + y)/2
print(average(4, 3))
```

Example:
```
print((lambda x, y: (x + y)/2)(4, 3))
```

Example:
```
def max(x,y):
    if x>y:
        return x
    else:
        return y
print(max(15,6))
```

Example:
```
max=lambda x,y: x if x>y else y
print(max(5,6))
```

Example:
```
print((lambda x,y: x if x>y else y)(2,3))
```

Example:
```
max=lambda x,y: x if x<y else y
print(max(5,6))
```

Lambda functions:
These are as follows:
1 map() 2 filter()   3 reduce()

We can pass function as argument to another function
Example:
```
map(function,sequence)
filter(function,sequence)
reduce(function,sequence)
```

map() function
Apply same function to each element of a sequence and return the modified list.

Syntax:
```
list=[m,n,p]
function(),f      ==> MAP  ==> New list,[f(m),f(n),f(p)]
```

Example:
```
n=[4,3,2,1]
print(list(map(lambda x:x**2,n)))
```

Example:
```
my_list = [1, 5, 4, 6, 8]
new_list = list(map(lambda x: x * 2 , my_list))
print(new_list)
```

Example:
```
num=[1.1,2.3,4.5]
Result=list(map(lambda x :int(x),num))
print(Result)
```

Example: WithOut Lambda:
```
PyTuple=(1,2,3,4,5)
def Compute(x):
    return x**2
PyTuple1=tuple(map(Compute,PyTuple))
print(PyTuple1)
```

filter() function: filter items out of a sequence, return filtered list

Syntax:
```
list,[m,n,p]
condition,c() ==> filter ==> New list [m,n]
                           if(m==condition)
```

Example:
```
n=[4,3,2,1]
print(list(filter(lambda x:x>2,n)))
```

Example:
```
my_list = [1, 2, 3, 5, 6, 8,10, 12,15]
Even_list = list(filter(lambda x: (x%2 == 0),my_list))
print(Even_list)
Odd_list = list(filter(lambda x: (x%2 != 0),my_list))
print(Odd_list)
```

Example: WithOut Lambda
```
def Even_Number(x):
    if x%2!=0:
        return True
    else:
        return False
PyList=[1,2,3,4,5,6,7,8,9,10]
PyList1=list(filter(Even_Number,PyList))
print(PyList1)
```

reduce() function:
1. Applies same operation to items of a sequence
2. Uses result of operation as first param of next operation
3. Returns an item, not a list, This function is defined in “functools” module.

Syntax:
```
list,[m,n,p]
function f() ==> reduce() ==> f(f(m,n),p)
```

Example:
```
import functools
n=[4,3,2,1]
print(functools.reduce(lambda x,y:x*y,n))
```

Example:
```
import functools
print(functools.reduce(lambda x,y:x+y,[1,2,3,4]))
result=sum([x for x in [1,2,3,4]])
print(result)
```

Python *args and **kwargs
In programming, we define a function to make a reusable code that performs similar operation. To perform that operation, we call a function with the specific value, this value is called a function argument in Python.

Example: Function to Sum of 2 numbers
```
def Add(x,y):
    print("Addition is: ",x+y)
    return()
Add(10,12)
```

```
Output: Addition is:  22
```

NOTE:
Lets see what happens when we pass more than 3 arguments in the Add() function.
TypeError: adder() takes 2 positional arguments but 3 were given

Introduction to *args and **kwargs in Python
In Python, we can pass a variable number of arguments to a function using special symbols. There are two special symbols:
1. *args (Non Keyword Arguments)
2. **kwargs (Keyword Arguments)

We use *args and **kwargs as an argument when we are unsure about the number of arguments to pass in the functions.

Python *args
As in the above example we are not sure about the number of arguments that can be passed to a function. Python has *args which allow us to pass the variable number of non keyword arguments to function. In the function, we should use an asterisk * before the parameter name to pass variable length arguments.

Example:
```
def Add(*num):
    Sum = 0
    for n in num:
        Sum=Sum+n
    print("Sum is:",Sum)
Add(3,5)
Add(4,5,6,7)
Add(1,2,3,5,6)
```

Understanding **kwargs
The double asterisk form of **kwargs is used to pass a keyworded, variable-length argument dictionary to a function. Again, the two asterisks (**) are the important element here, as the word kwargs is conventionally used, though not enforced by the language.

Example:
```
def print_kwargs(**kwargs):
        print(kwargs)
print_kwargs(kwargs1="KORA", kwargs2="Subba Raju")
print_kwargs(kwargs1="KORA", kwargs2="Subba Raju",kwargs3="20-Years")
```

Example:
```
def print_values(**kwargs):
    for key, value in kwargs.items():
        print("The value of {} is {}".format(key, value))
print_values(MyName="Raju", YourName="Ravi")
```

Example:
```
def print_values(**kwargs):
    for key, value in kwargs.items():
        print("The value of {} is {}".format(key, value))
print_values(
            Name1="Alex",
            Name2="Gray",
            Name3="Harper",
            Name4="Raju",
            Name5="SARA",
            Name6="SCOTT"
        )
```

PYTHON ITERATORS
An iterator is an object that contains a countable number of values. In Python, which implements the iterator protocol, which consist of the methods __iter__() and __next__().

Iterator vs Iterable
Lists, tuples, dictionaries, and sets are all iterable objects. They are iterable containers which you can get an iterator from. All these objects have a iter() method which is used to get an iterator.

Example:
```
for x in range(5):
    print(x)
```

Looping Through an Iterator
We can also use a for loop to iterate through an iterable object:

Example:
```
PyTuple = ("Apple", "Banana", "Cherry")
for item in PyTuple:
  print(item)
```

Example:
```
PyList=[1,2,3,4]
x=iter(PyList)
print(x)#just it retruns object memory location
print(next(x))
```

Example:
```
PyList=[1,2,3,4]
x=PyList.__iter__()
print(x)#just it retruns object memory location
print(x.__next__())
print(x.__next__())
```

Example:
```
PyTuple = ("Apple", "Banana", "Cherry")
PyIt = iter(PyTuple)
print(next(PyIt))
print(next(PyIt))
print(next(PyIt))
```

Strings are also iterable objects, containing a sequence of characters
Example:
```
Pystr="Banana"
PyIt= iter(Pystr)
print(next(PyIt))
print(next(PyIt))
print(next(PyIt))
print(next(PyIt))
print(next(PyIt))
```

Create an Iterator
The __iter__() method must always return the iterator object itself.
The __next__() method also allows you to do operations, must return the next item in the sequence.

Example:
```
class MyNumbers:
  def __iter__(self):
    self.a = 1
    return self

  def __next__(self):
    x = self.a
    self.a += 1
    return x

myclass = MyNumbers()
myiter = iter(myclass)
print(next(myiter))
print(next(myiter))
print(next(myiter))
print(next(myiter))
print(next(myiter))
```

StopIteration
The example above would continue forever if you had enough next() statements, or if it was used in a for loop. To prevent the iteration to go on forever, we can use the StopIteration statement.

In the __next__() method, we can add a terminating condition to raise an error if the iteration is done a specified number of times:

Example:
```
class MyNumbers:
  def __iter__(self):
    self.a = 1
    return self

  def __next__(self):
    if self.a <= 10:
      x = self.a
      self.a += 1
      return x
    else:
      raise StopIteration

myclass = MyNumbers()
myiter = iter(myclass)
for x in myiter:
  print(x)
```

Implementing Own Iterator Class:
Implementing Remote Control class that allows you to press next button to go to next channel.

Example:
```
class RemoteControl():
    def __init__(self):
        self.channels=['HBO','CNN','STAR','ABC','ESPN']
        self.index=-1
    def __iter__(self):
        return self
    def __next__(self):
        self.index+=1
        if self.index==len(self.channels):
            raise StopIteration
        return self.channels[self.index]
r=RemoteControl()
itr=iter(r)
print(next(itr))
print(next(itr))
print(next(itr))
print(next(itr))
print(next(itr))
```