```python
"""
Python Functions
Summary By: Varpe K.M.
Reference: https://www.geeksforgeeks.org/functions-in-python/
"""
"""
In Python, anonymous function means that a function
is without a name.

Python Lambdas are anonymous functions
which are small and restricted and
come without an identifier.


As we already know that def keyword is used to define
the normal functions and
the lambda keyword is used to create anonymous functions.
It has the following syntax:
            lambda arguments: expression

This function can have any number of arguments
but only one expression, which is evaluated and returned.

One is free to use lambda functions wherever
function objects are required.

You need to keep in your knowledge that lambda functions
are syntactically restricted to a single expression.

It has various uses in particular fields of programming
besides other types of expressions in functions.

Let's look at this example and try to understand
the difference between a normal def defined function and lambda function.

This is a program that returns the cube of a given value:
"""

# Python code to illustrate cube of a number
# showing difference between def() and lambda().
def cube(y):
    return y*y*y;

g = lambda x: x*x*x
print(g(7))

print(cube(5))

"""
Without using Lambda :

Here, both of them returns the cube of a given number.
But, while using def, we needed to define a function
with a name cube and needed to pass a value to it.
After execution, we also needed to return the result
from where the function was called using the return keyword.

Using Lambda :

Lambda definition does not include a “return” statement,
it always contains an expression which is returned.
```

```
We can also put a lambda definition anywhere a function is expected,
and we don’t have to assign it to a variable at all.
This is the simplicity of lambda functions.

example of a lambda function with an argument:
"""
(lambda z: z + 1)#(5)
_(5)  # most recent lambda fucntion will be called byusing this way
add = lambda x: x + 1
add(5)

"""
Lambda functions can be used along with built-in functions like
 filter(), map() and reduce().
"""
"""
Use of lambda() with filter()

The filter() function in Python takes in a function and a list as arguments.
This offers an elegant way to filter out all the elements
of a sequence “sequence”, for which the function returns True.
Here is a small program that returns the odd numbers from an input list:
"""

# Python code to illustrate
# filter() with lambda()
li = [5, 7, 22, 97, 54, 62, 77, 23, 73, 61]

final_list = list(filter(lambda x: (x%2 != 0) , li))

print(final_list)

"""
Output:
[5, 7, 97, 77, 23, 73, 61]
"""
"""
Use of lambda() with map()

The map() function in Python takes in a function and
a list as argument.

The function is called with a lambda function and a list and
 a new list is returned which
contains all the lambda modified items returned by
that function for each item. Example:
"""

# Python code to illustrate
# map() with lambda()
# to get double of a list.
li = [5, 7, 22, 97, 54, 62, 77, 23, 73, 61]

final_list = list(map(lambda x: x*2 , li))

print(final_list)
"""
Output:
[10, 14, 44, 194, 108, 124, 154, 46, 146, 122]
"""
"""
```

```python
Use of lambda() with reduce()

The reduce() function in Python takes in a function and
a list as argument.
The function is called with a lambda function and
a list and a new reduced result is returned.
This performs a repetitive operation over the pairs of the list.
This is a part of functools module. Example:
"""

# Python code to illustrate
# reduce() with lambda()
# to get sum of a list
from functools import reduce
li = [5, 8, 10, 20, 50, 100]
sum = reduce((lambda x, y: x + y), li)
print (sum)
"""
Output:
193

Here the results of previous two elements are added
to the next element and this goes on
till the end of the list like (((((5+8)+10)+20)+50)+100).

"""
```