Lambda (лямбда)

- Lambda - это анонимная функция (или по другому - выражение), которая не имеет имени и которую мы можем передать в качестве аргумента в другую функцию.
- Lambda - это однострочное простое выражение, в котором мы не выходим за пределы одной строки.

```
lambda arguments: expression
```

Фактически, лямбда выполняет некое выражение (какой-то питоновский код), и возвращает результат в то место, где эта лямбда была вызвана.

Чтобы понять всю пользу этой конструкции, попробуйте написать простой скрипт PySide, где есть кнопка, и попробуйте назначить на эту кнопку вызов функции с каким-либо аргументами. Например:

```
self.myButton.clicked.connect(self.someFunction(arg1, arg2))
```

У вас ничего не выйдет, поскольку в примере выше мы можем лишь указать имя функции, которую мы хотим назначить на кнопку. Мы не можем передать аргументы. Однако можем обойти это ограничение, используя Lambda.

Давайте рассмотрим простой пример, чтобы лучше понять, как работает lambda.

```
exp = lambda x: x * x
print exp( 13 ) # output: 26
```

В данном примере exp это просто переменная, которая в дальнейшем стала выражением. Если вы работали с выражениями в Maya, то это хорошая аналогия. На эту переменную мы назначали выражение, используя ключевое слово lambda, далее аргумент x (это можем быть любая переменная). Этот аргумент живет в пространстве этого выражения, далее, если мы где-то выполним print x - питон выдаст нам ошибку. После аргумента, поставив двоеточие :, мы пишем само выражение. Это выражение может использовать внешние переменные и функции. Мы не можем написать в лямбду полноценный скрипт, но некую формулу - легко.

Теперь переменная exp стала выражением, и поскольку она может принять один аргумент, выполнив print exp(13) выдаст результат в виде числа 26.

Рассмотрим более сложный пример. Допустим у нас есть некая функция, и мы хотим ею воспользоваться, но с некой поправкой. С помощью lambda это можно сделать следующим образом.

```
def Foo (val1, val2):
    if val2 < 0 :
        val2 *= -1 #make val2 positive
    return = val1 * val2

exp = lambda a = 1 , b = 1 , c= 1 : Foo(a,b) / float(c)
print exp( 2 , 3 , 10 ) # output: 0.6
```

Теперь рассмотрим пример выше, когда мы хотели назначить на кнопку вызов функции с аргументами.

```
  ...
   self.a = 14
   self.b = 17
   self.buttonRun.clicked.connect( lambda : self.test(self.a, self.b) )
   ...

def test (self, value_1 = 0 , value_2 = 0 ):
   print ( value_1 + value_2 )
```

Lamda отлично работает с такими конструкциями Python как Map, Filter, Reduce.

Map

Map - применяет функцию к каждому элементу итерируемого объекта (списка, словаря и т.д.)

```
def power2 (n):
    return n** 2

a = [ 1 , 2 , 3 , 4 , 5 , 6 ]
a_pow2 = map(power2, a)
print (list(a_pow2))
```

Часто вместо функции используется lambda.

```
a= [1,2,3,4,5,6]
a_pow2 = map(lambda x: x*x, a)
print (list(a_pow2))
```

В map() мы можем передать конкретную функцию, которая будет обрабатывать списки и возвращать результат. Например, интересный результат можно получить в примере ниже:

```
a = ["one", "two", "three"]
b = ["orange", "green", "blue"]
c = ["apple", "pineapple", "mellow"]

def foo(af, bf, cf):
  return "{} {} {}".format(af, bf, cf)

x = map(foo, a, b, c)
print (list(set(x)) )
['two green pineapple', 'three blue mellow', 'one orange apple']
```

Filter

filter() - просматривает каждый элемент списка и удаляет те, которые не соответствуют заданным критериям.

filter так же как и map использует функцию для фильтрации списка, но в отличие от map он может принимать только один входящий объект.

```
a = [1,2,-4,-5,6,-7,8,-9]
x = filter(lambda x: x > 0, a)
print sorted(list(set(a)))  #Out: [1,2,6,8]
```

Еще один пример:

```
def filt(x):
       if x > 0:
              return x

a = [1,2,-4,-5,6,-7,8,-9]
b = filter(filt, a)
print (list(b))
```

Если в качестве первого аргумента передать None (там где должна быть указана функция), то это отфильтрует входящий список, убрав все нулевые значения, такие как 0, False, None, [], "".