Python Workshop

Shivaji University, Kolhapur

Suresh Parit (https://www.linkedin.com/in/surajsigma/)

15-07-2018 ¶

Reference :

The History of Python by Guido van Rossum (http://python-history.blogspot.com/)

Python Course (https://www.python-course.eu/)

markdown-for-jupyter-notebooks (https://medium.com/ibm-data-science-experience/markdown-for-jupyter-notebooks-cheatsheet-386c05aeebed)

You know this

- What is Python
- Lists, Dictionaries, Tuples
- Arrays
- string functions
- control strucutres
- function and applications
- class , Objects and Inheritance

In [ ]:

We will discuss ...

- Lambda Functions
- Packages and Modules
  - Numpy
  - Scipy
  - Pandas
  - Matplotlib
- Programming Using Functions
- File Operations

LAMBDA FUNCTIONS

Basic syntax

lambda arguments : expression

lambda operator can have any number of arguments, but it can have only one expression. It cannot contain any statements and it returns a function object which can be assigned to any variable.

In [21]:
```
def add(x, y):
   return x + y

# Call the function
add(2, 3)  # Output: 5
```

Out[21]:
```
5
```

In [4]:
```
add= lambda x,y : x+y
```

In [5]:
```
add(2,4)
```

Out[5]:
```
6
```

What is type of add ?

In [6]:
```
type(add)
```

Out[6]:
```
function
```

MAP Function

Basic Syntax

map(function_object, iterable1, iterable2,...)

Map Functions expects a function object and any number of iterables like list, dictionary, etc. It executes the function_object for each element in the sequence and returns a list of the elements modified by the function object.

In [10]:
```
def sq(x):
  return x ** 2
```

Out[10]:
```
[1, 4, 9, 16]
```

In functional programming, functions transform input into output, without an intermediate representation of the current state.

In [16]:
```
input=[1,2,3,4]
output = []
for v in input:
    output.append(sq(v))
```

In [17]:
```
output
```

Out[17]:
```
[1, 4, 9, 16]
```

In [ ]:
```
list(map(sq, [1, 2, 3, 4])) # Output [1, 4, 9, 16]
```

similar function

In [15]:
```
list(sq(x) for x in [1,2,3,4])
```

Out[15]:
```
[1, 4, 9, 16]
```

First class functions : first class means that they can appear anywhere in a program, including return values and arguments of other functions.

In [18]:
```
sp_square=sq
```

In [19]:
```
sp_square(4)
```

Out[19]:
```
16
```

High-order functions are functions which can take functions as arguments (or return them as results).

In [22]:
```
add(sq(4),2)
```

Out[22]:
```
18
```

Reduce and Filter

Basic Syntax

filter(function, sequence)

offers an elegant way to filter out all the elements of a sequence "sequence", for which the function function returns True.

In [24]:
```
fibonacci = [0,1,1,2,3,5,8,13,21,34,55]
```

In [32]:
```
odd_numbers = list(filter(lambda x: x % 2==1, fibonacci))
odd_numbers
```

In [34]:
```
even_numbers = list(filter(lambda x: x % 2==0, fibonacci))
even_numbers
```

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

Basic Syntax

reduce(func, seq)

continually applies the function func() to the sequence seq. It returns a single value.

In [35]:
```
import functools
```

In [37]:
```
a=range(1,11)
```

In [38]:
```
a
```

Out[38]:
```
range(1, 11)
```

In [39]:
```
functools.reduce(lambda x,y: x+y, a)
```

Out[39]:
```
55
```

In [43]:
```
f1 = lambda a,b: a if (a > b) else b
```

In [41]:
```
from functools import reduce
```

In [44]:
```
reduce(f, [47,11,42,102,13])
```

Out[44]:
```
102
```

Some of the characteristics of Functional Programming are:

- Avoid state representation
- Data are immutable
- First class functions
- High-order functions
- Recursion

Try it ...

Imagine an accounting routine used in a book shop. It works on a list with sublists, which look like this:

Order Number | Book Title and Author | Quantity | Price per Item

34587 |Learning Python, Mark Lutz | 4 |40.95

98762 |Programming Python, Mark Lutz | 5 |56.80

77226 |Head First Python, Paul Barry | 3 |32.95

88112 |Einführung in Python3, Bernd Klein | 3 |24.99

Write a Python program, which returns a list with 2-tuples. Each tuple consists of a the order number and the product of the price per items and the quantity. The product should be increased by 10,- € if the value of the order is less than 100,00 €. Write a Python program using lambda and map.

In [46]:
```
orders = [ ["34587", "Learning Python, Mark Lutz", 4, 40.95],
           ["98762", "Programming Python, Mark Lutz", 5, 56.80],
           ["77226", "Head First Python, Paul Barry", 3,32.95],
           ["88112", "Einführung in Python3, Bernd Klein",  3, 24.99]]
```

In [47]:
```
orders
```

Out[47]:
```
[['34587', 'Learning Python, Mark Lutz', 4, 40.95],
 ['98762', 'Programming Python, Mark Lutz', 5, 56.8],
 ['77226', 'Head First Python, Paul Barry', 3, 32.95],
 ['88112', 'Einführung in Python3, Bernd Klein', 3, 24.99]]
```

In [ ]:
```
invoice_totals = list(map(lambda x: x if x[1] >= min_order else (x[0], x[1] + 10)
                          map(lambda x: (x[0],x[2] * x[3]), orders)))
```

In [52]:
```
k=list(map(lambda x : (x[0],x[2]*x[3]),orders))
```

In [54]:
```
k
```

Out[54]:
```
[('34587', 163.8),
 ('98762', 284.0),
 ('77226', 98.85000000000001),
 ('88112', 74.97)]
```

In [53]:
```
list(map(lambda x:x if x[1]>100 else (x[0],x[1]+10),k))
```

Out[53]:
```
[('34587', 163.8),
 ('98762', 284.0),
 ('77226', 108.85000000000001),
 ('88112', 84.97)]
```

Recursion: A recursive function calls itself on a smaller input, until the problem is reduced to some base case.

In [63]:
```
fact=lambda x: 1 if x==0 else x*fact(x-1)
```

In [65]:
```
fact(10)
```

Out[65]:
```
3628800
```

In [69]:
```
def fib(x):
    if x==0:
        return 0
    elif x==1:
        return 1
    else:
        return fib(x-1)+fib(x-2)
```

In [71]:
```
list(map(fib,[0,1,2,3,4,5,6]))
```

Out[71]:
```
[0, 1, 1, 2, 3, 5, 8]
```

In [72]:
```
fib1 = lambda n: n if n<=1 else fib1(n-1)+fib1(n-2)
```

In [74]:
```
list(map(fib,range(0,10)))
```

Out[74]:
```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

In [79]:
```
import datetime
```

In [ ]:
```

```