Anonymous Function

Lambda, map, filter and Reduce

Introduction

- Lambda is a tool for building functions, or more precisely, for building function objects. That means that Python has two tools for building functions: def and lambda.
- Features of Lambda
  - Actually, we don’t absolutely need lambda; we could get along without it. But there are certain situations where it makes writing code a bit easier
  - Normally, functions are created for one of two purposes:
    - (a) to reduce code duplication,
    - (b) to modularize code.

Syntax:

- *Basic Syntax*
  - *lambda arguments: expression*
- *list(map(lambda x: x * 2 , my_list))*
  - List→ Output Datatype
  - Map→ act like for loop

- *list(filter(lambda x: (x%2 == 0) , my_list))*
  - *Filter* →*act lik e if* condition

Example 1:

```
F= lambda x,y: x+y
F(4,5)
9
```

Example 2:

```
double=lambda x:x*2
 print(double(5))
10
double=lambda x:x*2
print(double)
```

Output : <function <lambda> at 0x0000000001CF3E18>

```
double=lambda x:x*2
x=2
print(double(x))
```

Map() function

- The map() function in Python takes in a function and a list.
- The function is called with all the items in the list and a new list is returned which contains items returned by that function for each item.
- Here is an example use of map() function to double all the items in a list.
- Example:
  - *my_list = [1, 5, 4, 6, 8, 11, 3, 12]*
  - *new_list = list(map(lambda x: x * 2 , my_list))*
  - *# Output: [2, 10, 8, 12, 16, 22, 6, 24]*
  - *print(new_list)*

Filter() function

- The filter() function in Python takes in a function and a list as arguments.
- The function is called with all the items in the list and a new list is returned which contains items for which the function evaluates to True.
- Here is an example use of filter() function to filter out only even numbers from a list.
- Filter Example:
  - *# Program to filter out only the even items from a list*
  - *my_list = [1, 5, 4, 6, 8, ’11’, 3, 12]*
  - *new_list = list(filter(lambda x: (x%2 == 0) , my_list))*
  - *# Output: [4, 6, 8, 12]*
  - *print(new_list)*

Reduce Function

- The reduce(fun,seq) function is used to apply a particular function passed in its argument to all of the list elements mentioned in the sequence passed along.
- This function is defined in “functools” module.
- Working :
    - At first step, first two elements of sequence are picked and the result is obtained.
    - Next step is to apply the same function to the previously attained result and the number just succeeding the second element and the result is again stored.
    - This process continues till no more elements are left in the container.
    - The final returned result is returned and printed on console.

Example

```
>>> import functools
>>> lis = [ 1 , 3, 5, 6, 2, ]
>>> print (functools.reduce(lambda a,b : a+b,lis))
17
>>> functools.reduce(lambda x,y: x+y, [47,11,42,13])
113
```

Powerful Utilities

List Comprehensive

Definition

- List comprehension is an elegant way to define and create list in Python. These lists have often the qualities of sets, but are not in all cases sets.
- List comprehension is a complete substitute for the lambda function as well as the functions map(), filter() and reduce(). For most people the syntax of list comprehension is easier to be grasped.
- Syntax:

[expression for item in list if conditional]

Normal Function vs List Comprehensive

Normal Function

```
for item in list:
    if conditional:
        expression
```

List Comprehensive

```
new_list = [expression(i) for i in old_list if
condition(i)]
```

Ex1

```
>>> Celsius = [39.2, 36.5, 37.3, 37.8]
>>> Fahrenheit = [ ((float(9)/5)*x + 32) for x in Celsius ]
>>> print(Fahrenheit)
[102.56, 97.700000000000003, 99.140000000000001,
100.03999999999999]
>>>
```

Ex 2:

- A Pythagorean triple consists of three positive integers a, b, and c, such that
- a2 + b2 = c2.
- Such a triple is commonly written (a, b, c), and the best known example is (3, 4, 5).
- The following list comprehension creates the Pythagorean triples:

>>> [(x,y,z) for x in range(1,30) for y in range(x,30) for z in range(y,30) if x**2 + y**2 == z**2]

Ex 3:

- Let A and B be two sets, the cross product (or Cartesian product) of A and B, written A×B, is the set of all pairs wherein the first element is a member of the set A and the second element is a member of the set B.
- Mathematical definition:
- A×B = {(a, b) : a belongs to A, b belongs to B}.
- It's easy to be accomplished in Python:

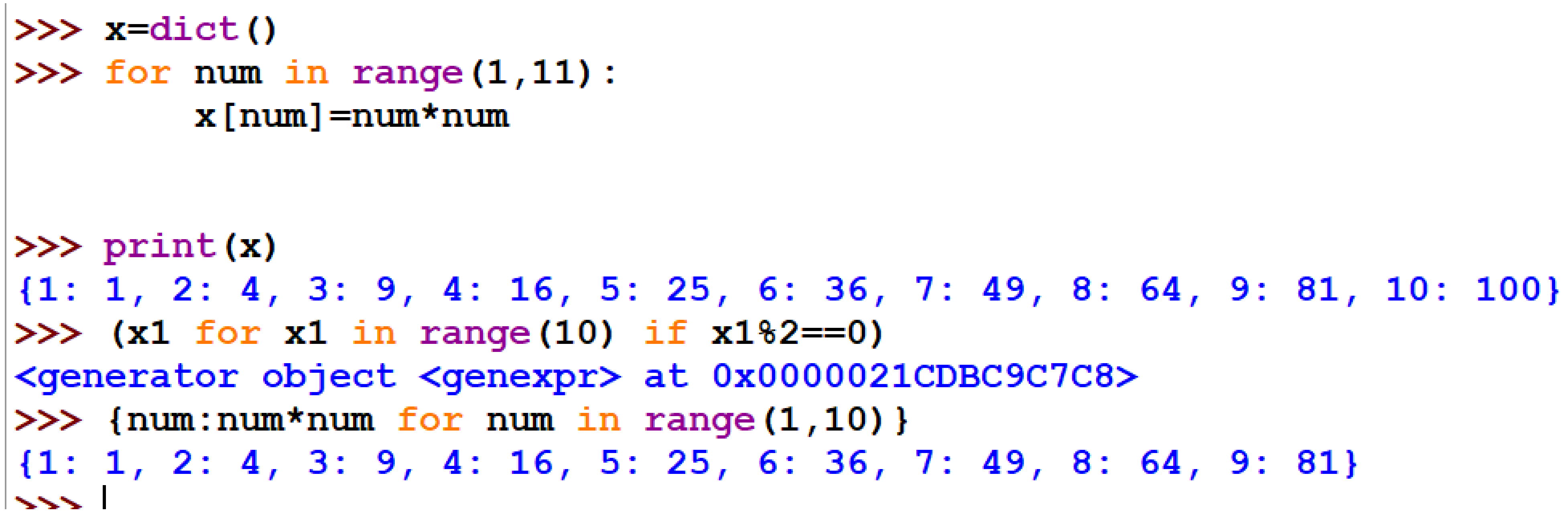
```
>>> colours = [ "red", "green", "yellow", "blue" ]
>>> things = [ "house", "car", "tree" ]
>>> coloured_things = [ (x,y) for x in colours for y in things ]
>>> print(coloured_things)
[('red', 'house'), ('red', 'car'), ('red', 'tree'), ('green', 'house'), ('green', 'car'), ('green', 'tree'), ('yellow', 'house'), ('yellow', 'car'), ('yellow', 'tree'), ('blue', 'house'), ('blue', 'car'), ('blue', 'tree')]
>>>
```

Generator Comprehension

- They are simply like a list comprehension but with parentheses - round brackets - instead of (square) brackets around it.
- Otherwise, the syntax and the way of working is like list comprehension, but a generator comprehension returns a generator instead of a list
- >>> x = (x **2 for x in range(20))
- >>> print(x)
- at 0xb7307aa4>
- >>> x = list(x)
- >>> print(x)
- [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361]

Dictionary Comprehensive

- The Comprehensive method with dictionary Example



```
>>> x=dict()
>>> for num in range(1,11):
        x[num]=num*num



>>> print(x)
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
>>> (x1 for x1 in range(10) if x1%2==0)
<generator object <genexpr> at 0x0000021CDBC9C7C8>
>>> {num:num*num for num in range(1,10)}
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}
>>> 
```

Great Job

Next Topic: Exception Handling