Advanced Functions

Introduction to Functional Programming

- Functional Programming (FP) is a programming paradigm.
- A Paradigm is a framework that indicates ways of computational thinking required to solve a given problem.
- Functional programming is a subset of declarative programming where functions are used to implement program logic for getting solutions.
- The core element of functional programming is a function rather than the statements to solve a given problem.
- Functional programming starts with splitting the problem into smaller subproblems.

- The notion of “Pure functions” is unique in functional programming.
- The main requirement of functional programming is that the function should not have side effects or functions with minimal side effects.
- Functional programming does not maintain state information. There are two important concepts: State and Mutation.
- The function should produce the same result given the same inputs making functionals deterministic.

Characteristics of functional programming

- Use only functions.
- Combine functions to form first-class or higher-order functions.
- Usage of recursion instead of iteration.
- No state information or side effects.
- The order of functional calls is not relevant.
- Pure functions, where pure functions are defined as functions that have no side effects.
- Iterators over lists.

Functional Programming

Advantages

- Functional programming creates less code with much clarity and unambiguity.
- Mathematically provable.
- Multiprocessing can be done.

Disadvantages

- Learning is a bit difficult as codes are very compact.
- Not all functions are pure functions.
- Recursion is inefficient, and hence FP can be in efficient at times.

Functions as Objects

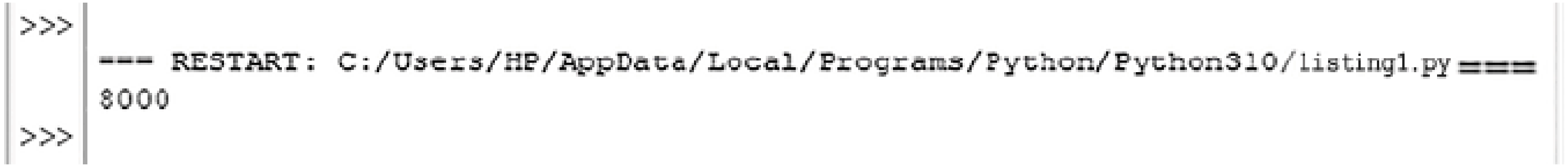
- Functions are first-class citizens in functional programming. The terminologies of first-class citizens and higher-order functions are interchangeable.
- First-class citizens refer to the tight integration of functions with the programming language. Such as,
- A function can be stored in a variable.
- Function can be passed as a parameter.
- Functions can be used to manipulate collections.
- Manipulated nay data types using functions.

Illustration of a function stored in a variable

```
def cube(x):
    return x*x*x
y = cube(20)   # Functions are stored in a variable
print(y)
```

The above code is stored as listing.py and executed in Interactive mode.

output of this code:



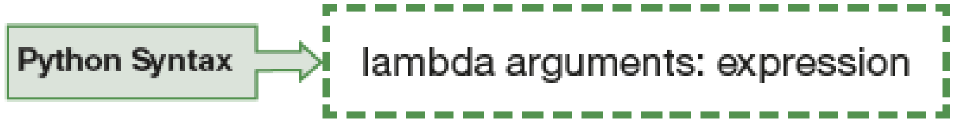
```
>>>
    === RESTART: C:/Users/HP/AppData/Local/Programs/Python/Python310/listing1.py ===
    8000
>>>
```

There can be alias on functions. An alias is a dummy name. The following example shows the alias where both s and t refer to the same object.

Lambda function

- Lambda function is a restricted, one-line function in python.
- Lambda are anonymous or name-less functions, as they have no name, unlike ordinary functions.
- Lambda calculus is a cross of functional abstraction and application of the function.
- In a way, lambda calculus promotes,
- Usage of functions only.
- No state or side effects.
- The order of evaluation is irrelevant.

Syntax of lambda function in Python



Python's anonymous functions must have three components to work: viz., lambda as a keyword, parameters (or bound variables), and function body.

- lambda: to create a lambda function.
- parameters: a lambda function might have a single or several parameters. Only commas separated arguments following the lambda keyword are required.
- expression: this is the main part of the function. This is where you specify the operation performed by the function. A lambda function can have only one expression.

Example: write a lambda function to return a cube of a number

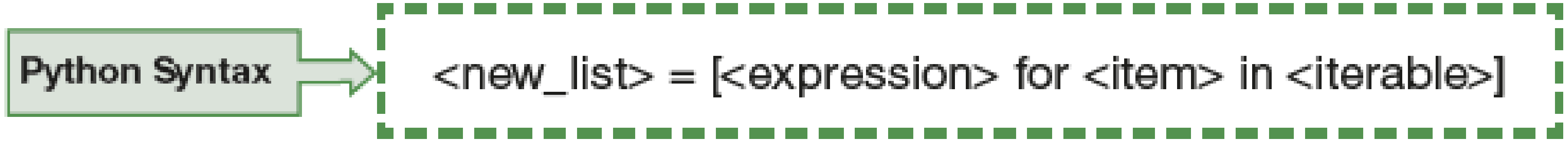
```
>>> x = lambda a: a**3

>>> print(x(4))

64

>>>
```

List Comprehensions

- There are three key Python tools used to process static sequences and streams. These elegant tools are used to process sequences and streams. The tools are listed below.

1. List Comprehensions
2. Iterators
3. Generators

The Syntax of list comprehension is given as follows.



Python Syntax

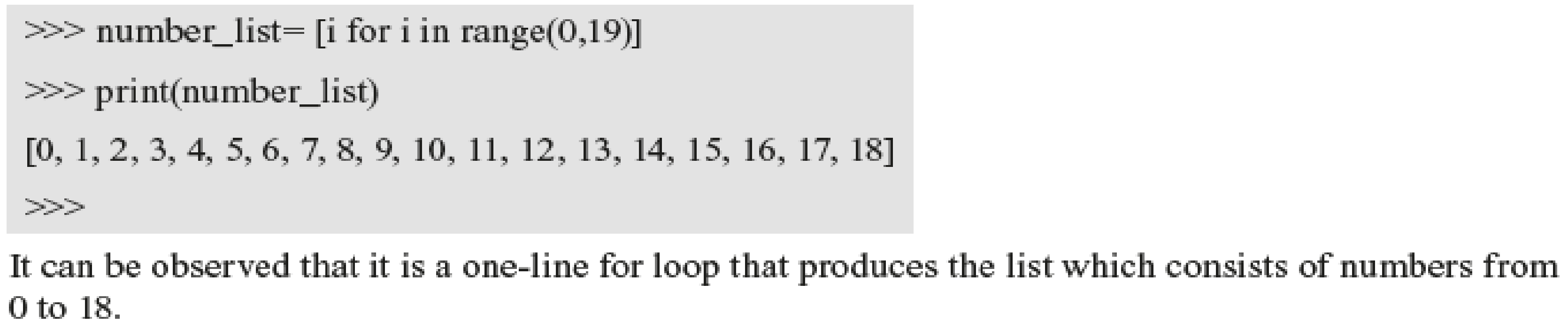
```
<new_list> = [<expression> for <item> in <iterable>]
```

The expression can be any valid Python expression, and the iterate can be any object like,

1. List
2. Strings
3. Tuple
4. Dictionary

List comprehensions provide immense documentation value as list comprehensions are self-documentary.

Example: Generate a list of numbers from 1 to 20. The numbers can be generated traditionally using a range function.

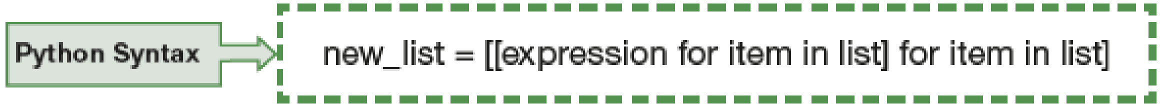


```
>>> number_list= [i for i in range(0,19)]
>>> print(number_list)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]
>>>
```

It can be observed that it is a one-line for loop that produces the list which consists of numbers from 0 to 18.

Nested list comprehension

- A list comprehension inside another list comprehension in analogous to nested for loops. Nested loops are implemented in the following way.

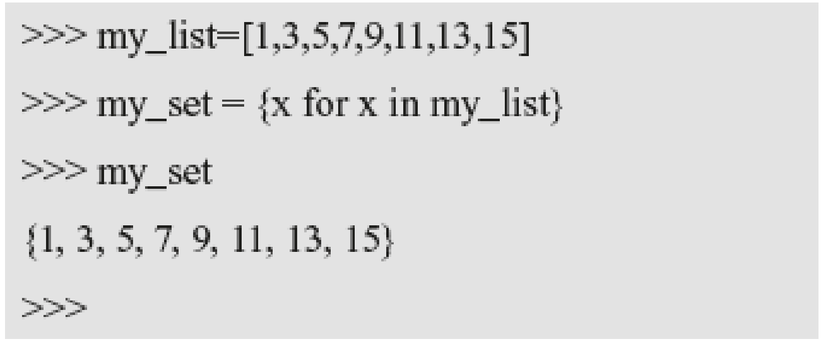


Python Syntax

```
new_list = [[expression for item in list] for item in list]
```

- Set Comprehensions

Python supports set comprehensions similar to list comprehensions.



```
>>> my_list=[1,3,5,7,9,11,13,15]
>>> my_set = {x for x in my_list}
>>> my_set
{1, 3, 5, 7, 9, 11, 13, 15}
>>>
```

Dictionary Comprehensions

Dictionary Comprehensions also used curly braces.

Dictionary comprehension does not directly operate on the dictionaries but with the items() couples with tuples to manipulate values and keys.

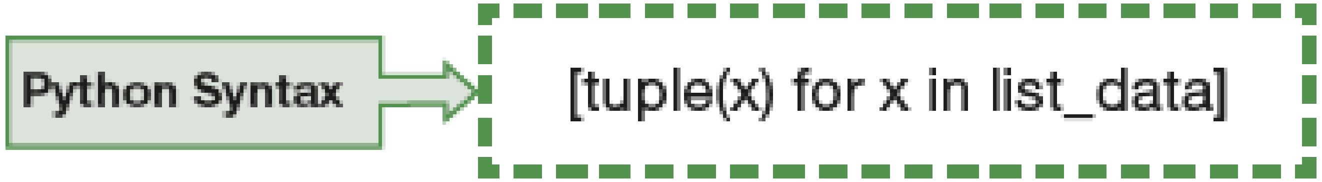
Dictionary Comprehension can be used to swap the keys and values in a dictionary.

List Comprehension in Tuples

Sequences are represented by parenthesis() and commas in a tuple.

A tuple is an ordered list.

Among the several ways to create a list of tuples, one method is by using the list comprehension and tuple() method.



```
[tuple(x) for x in list_data]
```

List Comprehension with Sequence Processing

- One can perform many computations on sequences. The ability to pass a parameter and receive it makes such functions higher-order functions.
- Some of the helpful higher-order functions are
- Map
- Filter
- Reduce

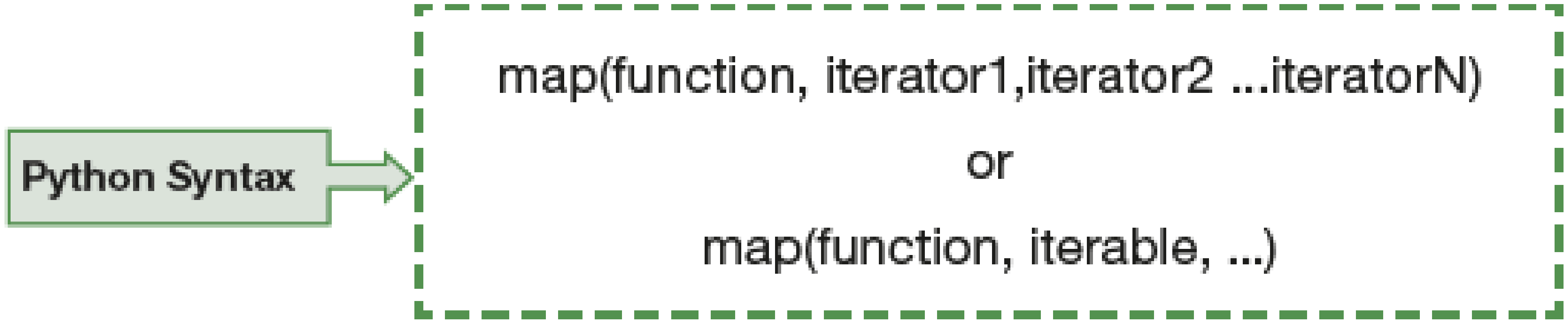
Map Function

- Map() is a built-in function in Python.
- It works as a function on all the items of a supplied iterable (list, tuple, or set).
- The elements of a python map object are looped through since it is In iterator.
- The map function that is used to apply every element of an iterable object.
- The map() function takes two arguments.

1) Any python functions
2) Iterable Object

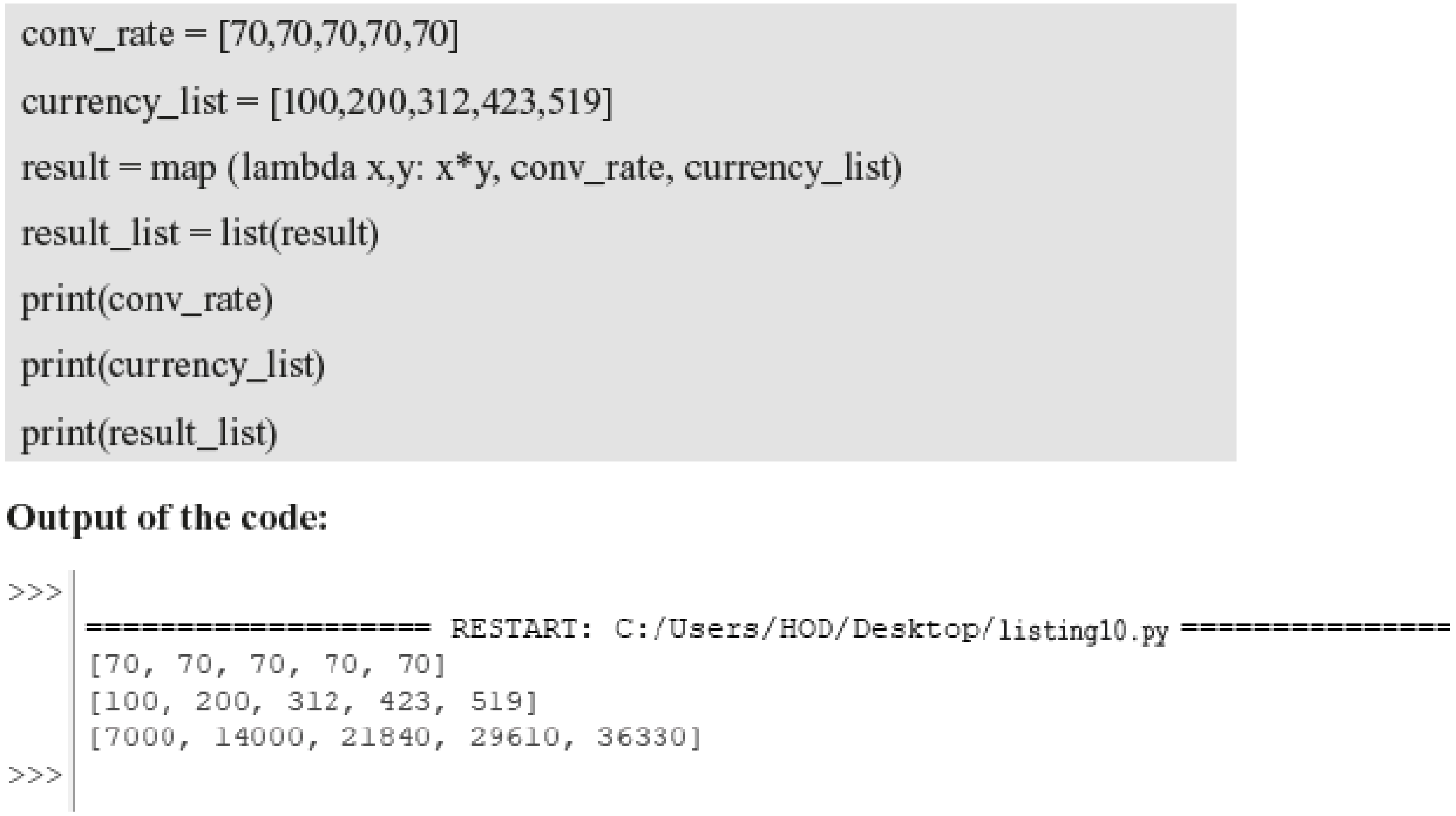
Map function

- In Python, iterables are lists, tuples, dictionaries, or sets.
- The function transforms the elements of the iterable object to give the resultant.
- The syntax of the map() function is given as follows,



```
map(function, iterator1,iterator2 ...iteratorN)
or
map(function, iterable, ...)
```

Example: To illustrate the change of case using the Map function



```
conv_rate = [70,70,70,70,70]
currency_list = [100,200,312,423,519]
result = map (lambda x,y: x*y, conv_rate, currency_list)
result_list = list(result)
print(conv_rate)
print(currency_list)
print(result_list)
```

Output of the code:

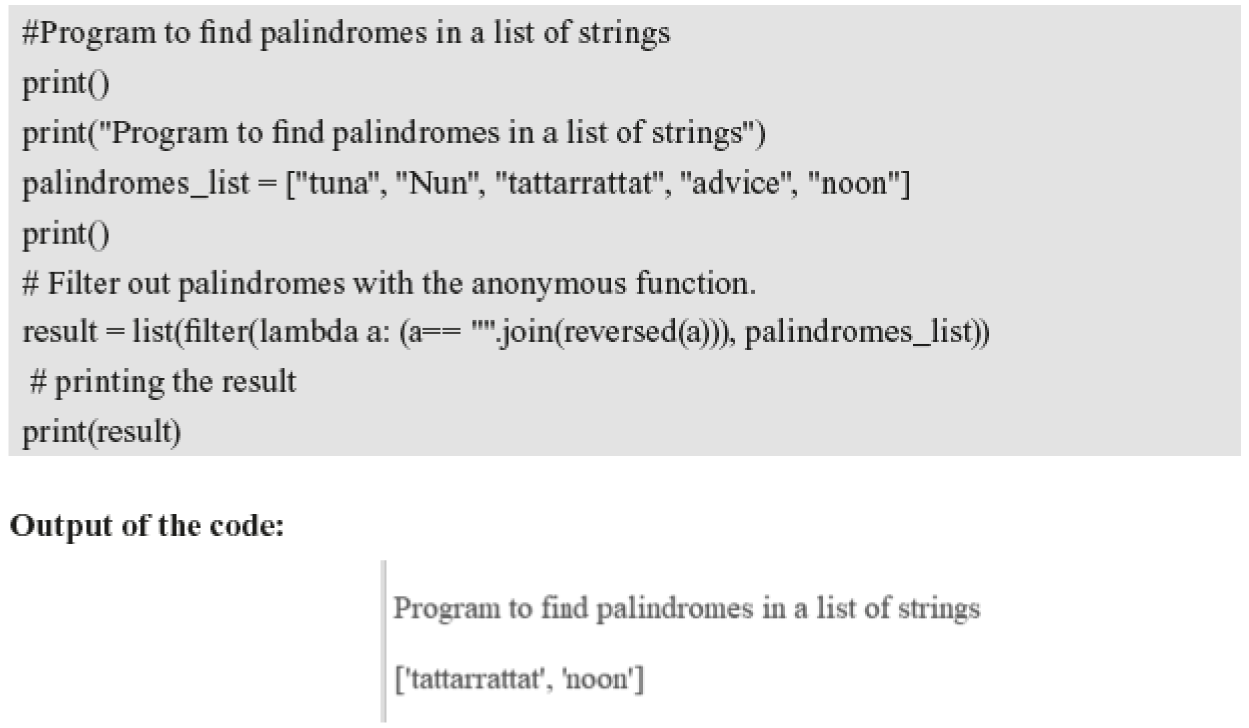
```
>>>
    =================== RESTART: C:/Users/HOD/Desktop/listing10.py ===============
    [70, 70, 70, 70, 70]
    [100, 200, 312, 423, 519]
    [7000, 14000, 21840, 29610, 36330]
>>>
```

Filter Function

- The function filter() takes two inputs

1. Iterable object
2. A condition

Example: To illustrate the change of case using the filter function



```
#Program to find palindromes in a list of strings
print()
print("Program to find palindromes in a list of strings")
palindromes_list = ["tuna", "Nun", "tattarrattat", "advice", "noon"]
print()
# Filter out palindromes with the anonymous function.
result = list(filter(lambda a: (a== "".join(reversed(a))), palindromes_list))
 # printing the result
print(result)
```

Output of the code:

```
Program to find palindromes in a list of strings

['tattarrattat', 'noon']
```

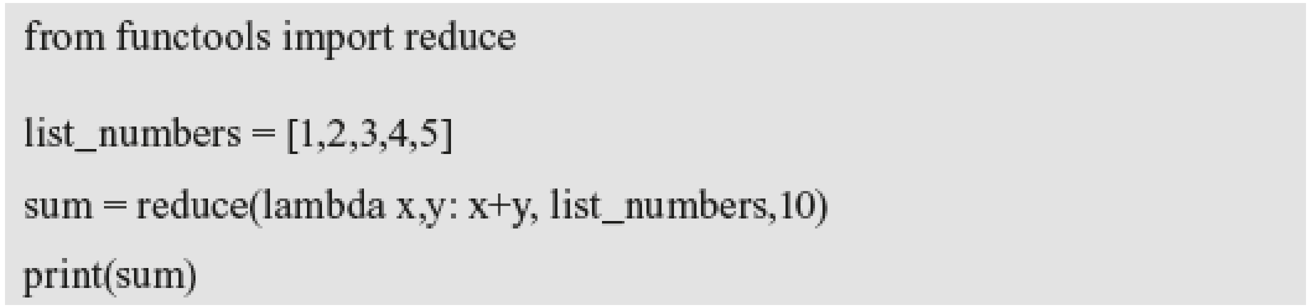
Reduce Function

- Reduce function reduces a set of values to a single number.
- For example,, given set of numbers, reduce function can reduce to a sum.

Reduce function has two arguments.

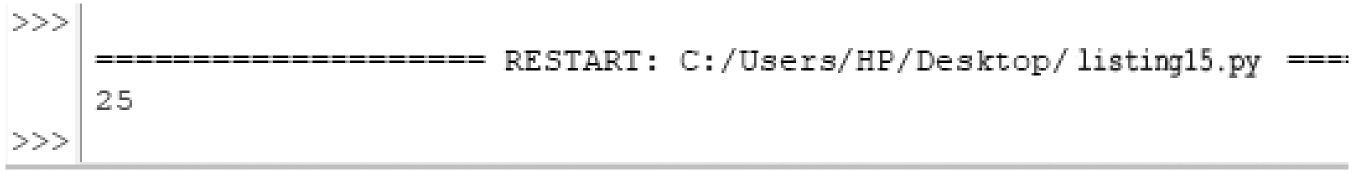
1. A function
2. A Sequence

reduce() apply a function over the series of values and returns a single value.



```
from functools import reduce

list_numbers = [1,2,3,4,5]
sum = reduce(lambda x,y: x+y, list_numbers,10)
print(sum)
```

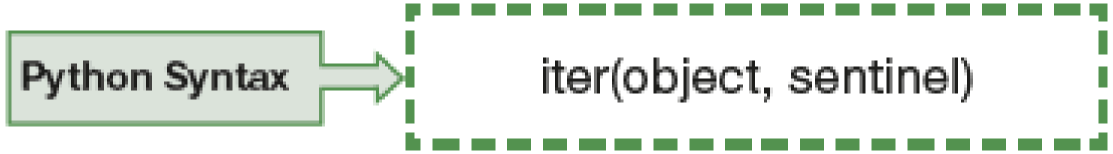
Output of the code:



```
>>>
    ==================== RESTART: C:/Users/HP/Desktop/ listing15.py  ===
    25
>>>
```

Iterators

- for-loops are used for iteration with objects, such as lists, sets, etc. by default it would iterate over the whole sequences without user significant control.
- The control of this iteration is brought by iterators.
- The iterators do provide an iterator protocol. It is a protocol that has two methods, __iter__(or iter()) and __next__(or nect())
- The syntax of iter() is given as:


Python Syntax

```
iter(object, sentinel)
```

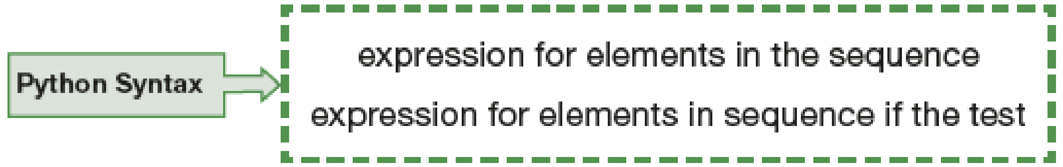
Generators

- Generators are special functions that helps to define user-defined iterators.
- The generator is a construct that generated a dynamic sequences and it generated an iterator.
- A generator is a function that emits one values at a time when called upon.
- The difference between generator and list comprehensions is that

1. There is no memory cost involved in the generator.
2. These generators do not compute until they are needed. This is called Lazy Evaluation.

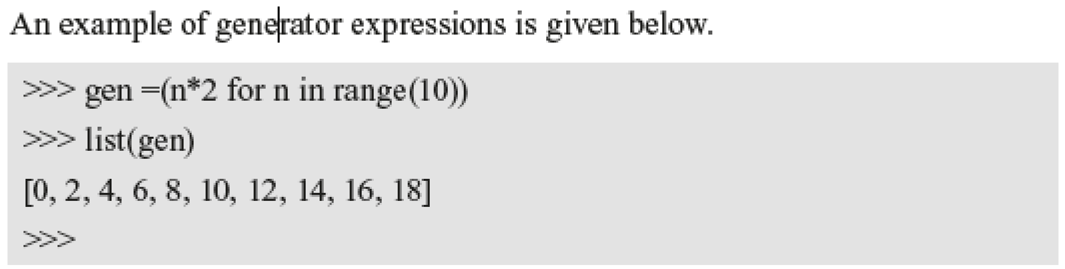
Generator Expressions

- Generator expressions are the product of list comprehensions and generator functions.
- It produces a generator as output lazily as it used the concept of lazy evaluation.
- The syntax of generator expressions is given below.



expression for elements in the sequence

expression for elements in sequence if the test



An example of generator expressions is given below.

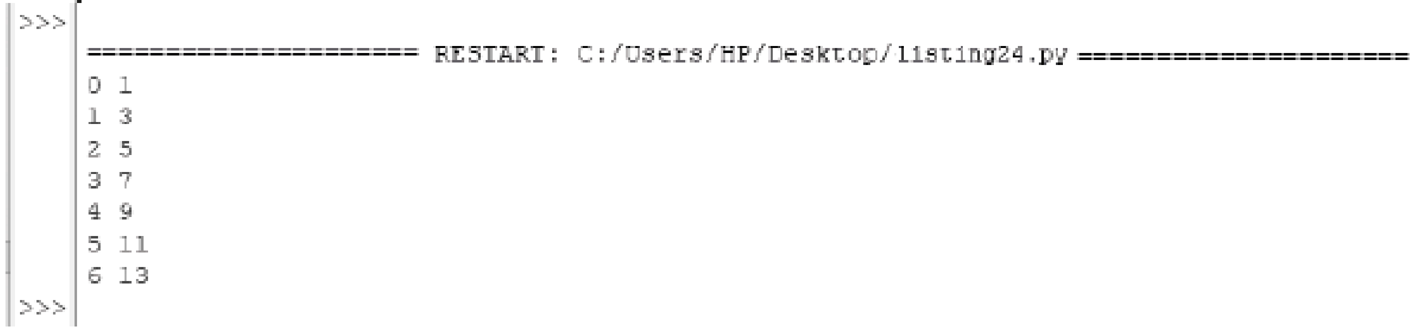
```
>>> gen =(n*2 for n in range(10))
>>> list(gen)
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
>>>
```

Itertools, Special Iterators, and Generators

- Itertools is a module.
- It provides many useful functions and generators with a wealth of built-in functions, such as enum() and sum().
- The enumerate() method keeps track of the elements included in a list, tuple, set or other iterable data structure.
- Enumerate allows to track the item along with the index.

```
L = {1, 3, 5, 7, 9, 11, 13}
for i, val in enumerate(L):
        print (i,val)
```

Output of the code:



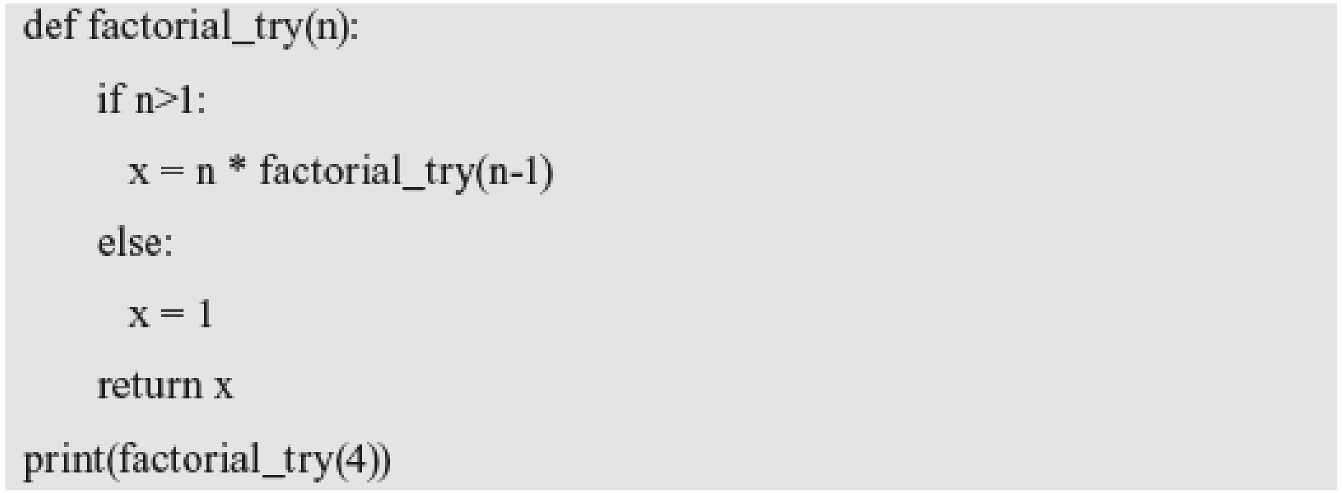
```
>>>
    ==================== RESTART: C:/Users/HP/Desktop/listing24.py ====================
    0 1
    1 3
    2 5
    3 7
    4 9
    5 11
    6 13
>>>
```

Recursion

- Functional programming prefer recursion over iteration.
- Loops maintain the states and involve mutable variables.
- Recursion is a useful functional programming technique that solves the problem using a technique called problem reduction.
- A function is called recursive because it calls by itself. A recursive program is one where recursive functions are used as an alternative for iteration.
- Recursive algorithms have two parts.

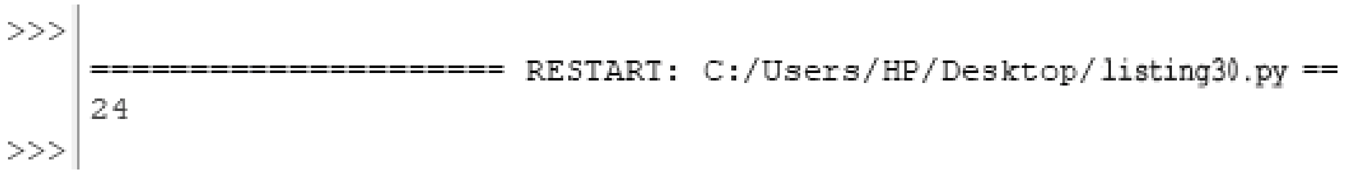
1. First part of recursive program is one or more base cases. Base cases provide trivial solutions and serve as termination conditions, The base problem is solved directly.
2. The second part is an inductive step. This step reduces the problem to a smaller version. This is called reducing step.

Example: Recursive program to find the factorial of a number.



```
def factorial_try(n):
    if n>1:
        x = n * factorial_try(n-1)
    else:
        x = 1
    return x
print(factorial_try(4))
```

Output of the code:



```
>>>
    ==================== RESTART: C:/Users/HP/Desktop/listing30.py ==
    24
>>>
```

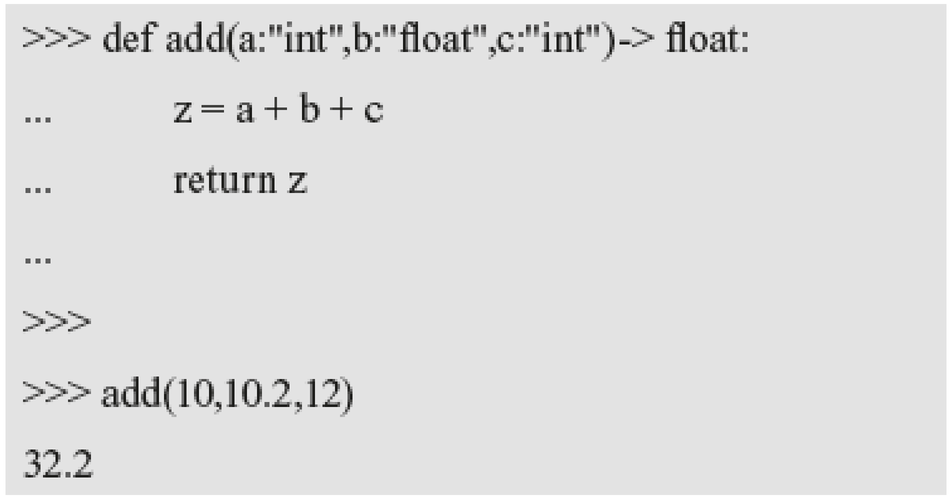
Closed Functions and Function Annotators or Decorators

- One can add a function into another function.
- This function is known as an inner function or embedded function.
- The other function is known as the outer function. The advantage of the inner function is that it can have access to the variables that are defined and available in the outer function.

1. The inner function cannot be accessed by the outside function.
2. The outer function can return the reference of the inner function. Using the reference , one can execute the inner function.
3. An inner function can serve as a helper function that can help another function.

Function Annotators

- These are meta-information that given for the benefit of users especially third-party python packages.
- These annotations are executed only in compline time and not used for run time
- Example of functional annotator.



```
>>> def add(a:"int",b:"float",c:"int")-> float:
...         z = a + b + c
...         return z
...
>>>
>>> add(10,10.2,12)
32.2
```