python for Computational Problem Solving - pCPS - Functional_Programming_Testing Lecture Slides - Class #39_#40

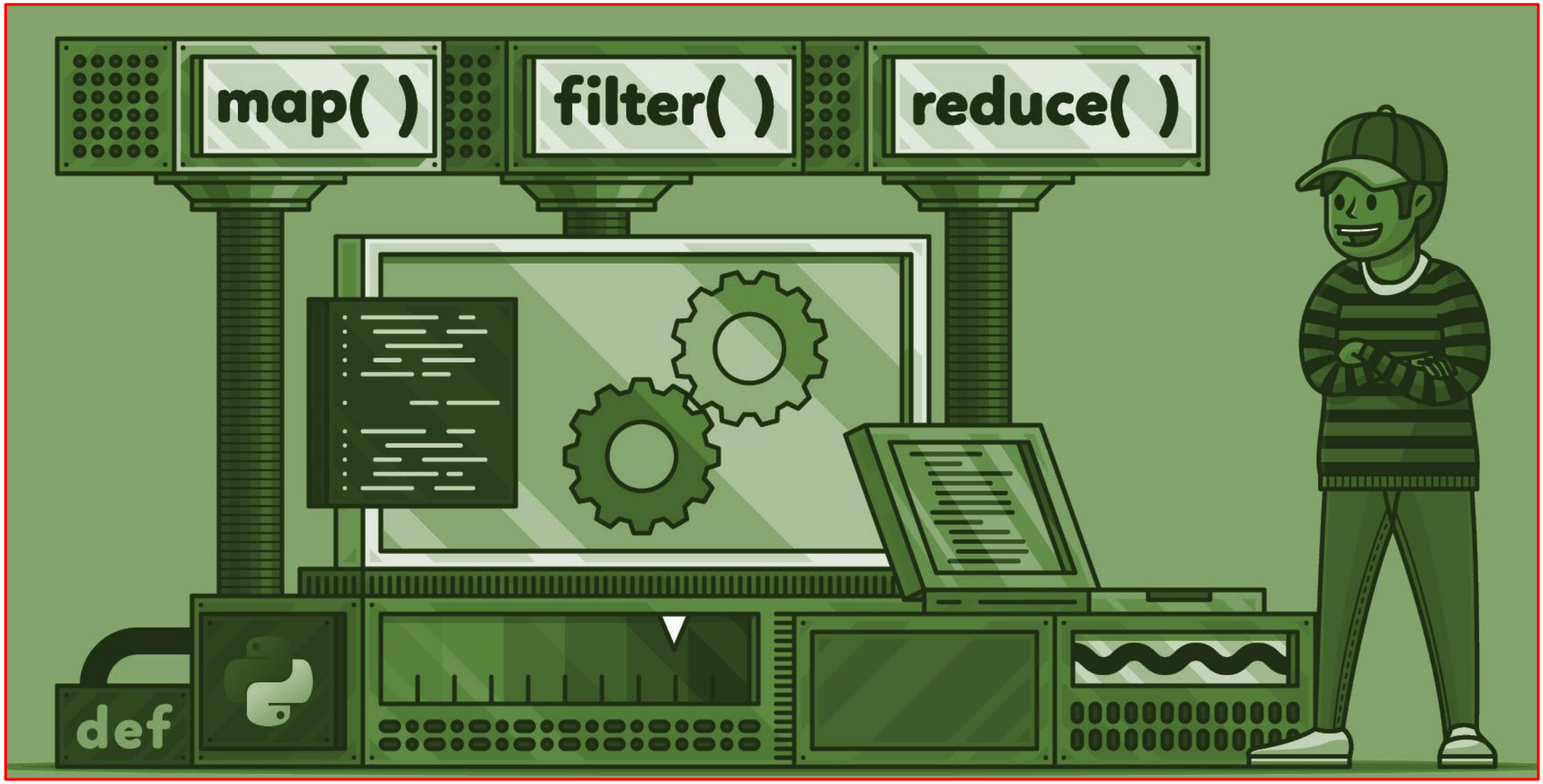
Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University

python for Computational Problem Solving Syllabus

Unit IV: Functional Programming, Modules,Testing and Debugging - 10 Hours

- Functional Programming - map, filter, reduce, max, min, lambda function
- list comprehension,
- Modules - import mechanisms
- Testing
  - Pytest , Function testing with Doctest
  - pdb debugger commands.

Functional Programming in python



Applying a Function to an Iterable With map() in python

- The first function on the docket is **map()**, which is a python built-in function.
- With **map()**, you can apply a function to each element in an **iterable** in turn, and map() will **return** an **iterator** that yields the results.
- This can allow for some very concise code because a map() statement can often take the place of an explicit loop
- The syntax for calling map() on a single iterable
  - **map(<f>, <iterable>)**
- **map(<f>, <iterable>)** returns in **iterator** that **yields** the **results** of applying **function <f>** to each **element** of **<iterable>**
- It **returns** an **iterator** called a **map object**.

```
def Ulta(S):
    return(S[::-1])

Names = ['PESU','First','Semester', 'PSection','Aug-Mar','2021-2022']

UltaS = map(Ulta,Names)

print('List-->',Names)
print(type(UltaS))
print(UltaS)
print('Ulta-->',[Individual for Individual in UltaS ])
```

```
List--> ['PESU', 'First', 'Semester', 'PSection', 'Aug-Mar', '2021-2022']
<class 'map'>
<map object at 0x7fe0724e1400>
Ulta--> ['USEP', 'tsriF', 'retsemeS', 'noitceSP', 'raM-guA', '2202-1202']
```

Applying a Function to an Iterable With map() in python

- The first function on the docket is **map()**, which is a python built-in function.
- With **map()**, you can apply a function to each element in an **iterable** in turn, and map() will **return** an **iterator** that yields the results.
- This can allow for some very concise code because a map() statement can often take the place of an explicit loop
- The syntax for calling map() on a single iterable
  - **map(<f>, <iterable>)**
- **map(<f>, <iterable>)** returns in **iterator** that **yields** the **results** of applying **function <f>** to each **element** of **<iterable>**
- It **returns** an **iterator** called a **map object**.

```
Names = ['PESU','First','Semester', 'PSection','Aug-Mar','2021-2022']

UltaS = map(lambda S:S[::-1],Names)
print('List-->',Names)
print(type(UltaS))
print(UltaS)
print('Ulta-->',list(UltaS))
```

```
List--> ['PESU', 'First', 'Semester', 'PSection', 'Aug-Mar', '2021-2022']
<class 'map'>
<map object at 0x7fe07234b9a0>
Ulta--> ['USEP', 'tsriF', 'retsemeS', 'noitceSP', 'raM-guA', '2202-1202']
```

Applying a Function to an Iterable With map() in python

- The first function on the docket is **map()**, which is a python built-in function.
- With **map()**, you can apply a function to each element in an **iterable** in turn, and map() will **return** an **iterator** that yields the results.
- This can allow for some very concise code because a map() statement can often take the place of an explicit loop
- The syntax for calling map() on a single iterable
  - **map(<f>, <iterable>)**
- **map(<f>, <iterable>)** returns in **iterator** that **yields** the **results** of applying **function <f>** to each **element** of **<iterable>**
- It **returns** an **iterator** called a **map object**.

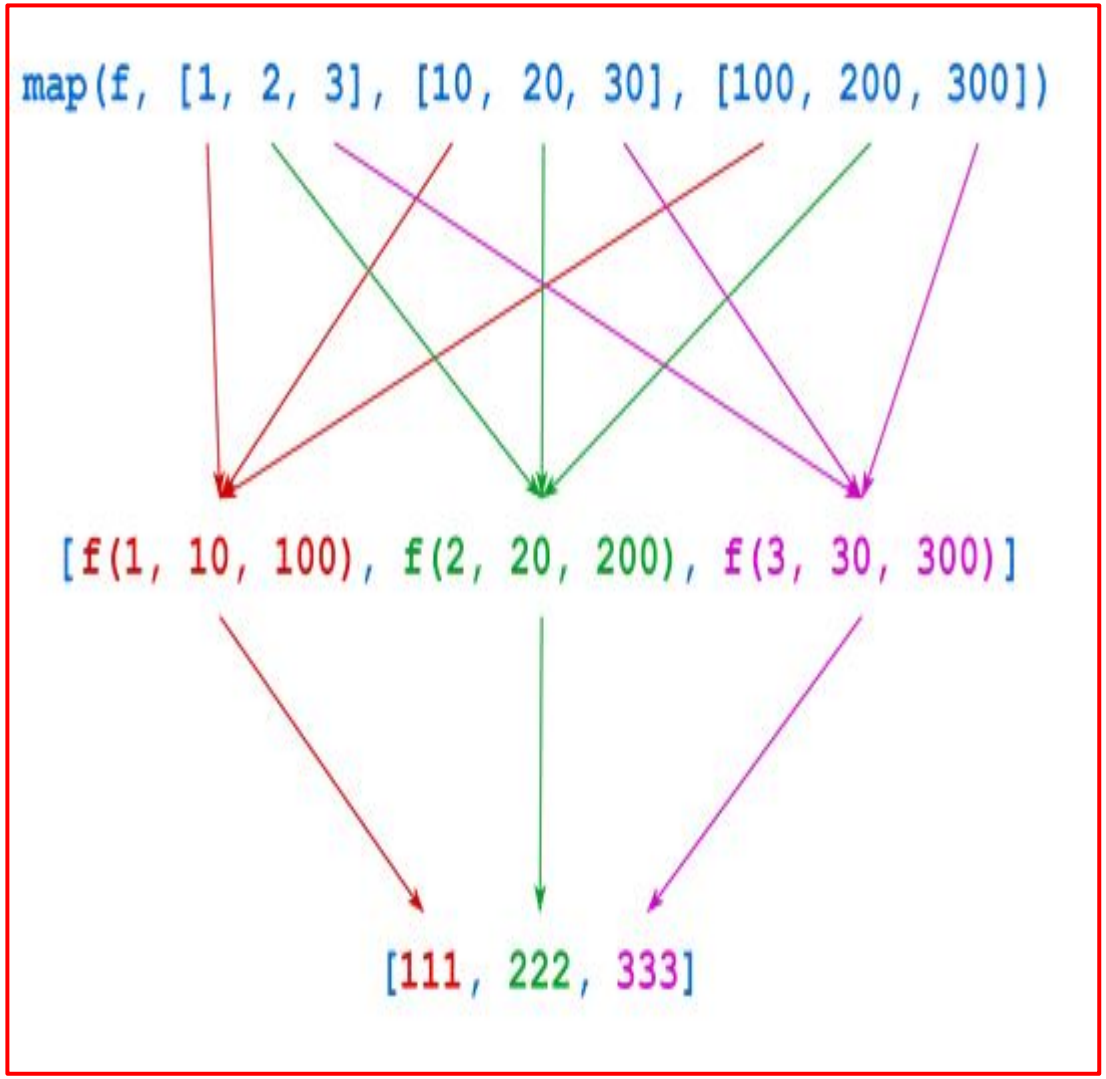
```
Names = ['PESU','First','Semester', 'PSection','Aug-Mar','2021-2022',8.2]

PNames = list(map(str,Names))
UltaS = list(map(lambda S:S[::-1],PNames))
print('List-->',Names)
print('PNames-->',PNames)
print(type(UltaS))
print(UltaS)
print('Ulta-->',list(UltaS))
```

```
List--> ['PESU', 'First', 'Semester', 'PSection', 'Aug-Mar', '2021-2022', 8.2]
PNames--> ['PESU', 'First', 'Semester', 'PSection', 'Aug-Mar', '2021-2022', '8.2']
<class 'list'>
['USEP', 'tsriF', 'retsemeS', 'noitceSP', 'raM-guA', '2202-1202', '2.8']
Ulta--> ['USEP', 'tsriF', 'retsemeS', 'noitceSP', 'raM-guA', '2202-1202', '2.8']
```

Applying a Function to an Iterable With map() in python

- There's **another form** of **map()** that takes more than one iterable argument
  - **map(<f>, <iterable$_1$>, <iterable$_2$>, ..., <iterable□>)**
- **map**(<f>, <iterable1>, <iterable2>, ..., <iterablen>) **applies** <f> to the **elements** in **each <iterablei>** in **parallel** and returns an **iterator** that **yields** the **results**
- The **number** of <iterablei> arguments specified to map() must **match** the **number** of **arguments** that **<f> expects**.
- **<f>** acts on the **first** item of **each <iterablei>**, and that **result** becomes the **first item** that the **return iterator** yields.
- **Then <f> acts** on the **second** item in **each <iterablei>**, and that **becomes** the **second yielded item**, and **so on**



Applying a Function to an Iterable With map() in python

- There's **another form** of **map()** that takes more than one iterable argument
    - **map(<f>, <iterable$_1$>, <iterable$_2$>, ..., <iterable□>)**
- **map**(<f>, <iterable1>, <iterable2>, ..., <iterablen>) **applies** <f> to the **elements** in **each <iterablei>** in **parallel** and returns an **iterator** that **yields** the **results**
- The **number** of <iterablei> arguments specified to map() must **match** the **number** of **arguments** that **<f> expects**.
- **<f>** acts on the **first** item of **each <iterablei>**, and that **result** becomes the **first item** that the **return iterator** yields.
- **Then <f> acts** on the **second** item in **each <iterablei>**, and that **becomes** the **second yielded item**, and **so on**

```
def f(a,b,c):
    return a+b+c

AList = map(f,[1,2,3],[4,5,6],[7,8,9])
print(list(AList))
```

```
[12, 15, 18]
```

```
AList = map(
            (lambda a,b,c:a+b+c),
            [1,2,3],
            [4,5,6],
            [7,8,9]
        )

print(list(AList))
```

```
[12, 15, 18]
```

Selecting Elements From an Iterable With filter() in python

- **filter()** allows you to select or filter items from an iterable based on evaluation of the given function.
    - **filter(<f>, <iterable>)**
- **filter(<f>, <iterable>)** applies **function** <f> to **each element** of <iterable> and **returns** an **iterator** that **yields all items** for which **<f>** is **truthy**. Conversely, it **filters out** all **items** for which <f> is **falsy**.
- **Many objects** and **expressions** are **not** equal to **True** or **False**.
- **Nonetheless**, they **may** still be **evaluated** in **Boolean context** and **determined** to be **"truthy"** or **"falsy."**
- The Boolean value **False**
    - Any value that is numerically zero (0, 0.0, 0.0+0.0j)
    - An empty string
    - An object of a built-in composite data type which is empty (see below)
    - The special value denoted by the Python keyword None
- Virtually **any** other **object built** into **Python** is regarded as **True**
- One can determine the **"truthiness"** of an **object** or **expression** with the built-in **bool()** function.
- **bool()** returns **True** if its argument is **truthy** and **False** if it is **falsy**

```
print('Truthy')
print(bool(True))
print(bool(1))
print(bool(1.5))
print(bool(1+1j))
print(bool(' '))
print(bool([0]))
print(bool((0,0)))
print(bool({0:''}))
```

```
Truthy
True
True
True
True
True
True
True
True
True
```

```
print('Falsy')
print(bool(False))
print(bool(0))
print(bool(0.0))
print(bool(0+0j))
print(bool(''))
print(bool([]))
print(bool(()))
print(bool({}))
print(bool(None))
```

```
Falsy
False
False
False
False
False
False
False
False
False
False
```

Selecting Elements From an Iterable With filter() in python

- **filter()** allows you to select or filter items from an iterable based on evaluation of the given function.
    - **filter(<f>, <iterable>)**
- **filter(<f>, <iterable>)** applies **function** <f> to **each element** of <iterable> and **returns** an **iterator** that **yields all items** for which **<f>** is **truthy**. Conversely, it **filters out** all **items** for which <f> is **falsy**.
- **Many objects** and **expressions** are **not** equal to **True** or **False**.
- **Nonetheless**, they **may** still be **evaluated** in **Boolean context** and **determined** to be **"truthy"** or **"falsy."**
- The Boolean value **False**
    - Any value that is numerically zero (0, 0.0, 0.0+0.0j)
    - An empty string
    - An object of a built-in composite data type which is empty (see below)
    - The special value denoted by the Python keyword None
- Virtually **any** other **object built** into **Python** is regarded as **True**
- One can determine the **"truthiness"** of an **object** or **expression** with the built-in **bool()** function.
- **bool()** returns **True** if its argument is **truthy** and **False** if it is **falsy**

```
def NonEmpty(S):
    return S>''

print(list(filter(NonEmpty,['1','2','3','',' ','','PESU'])))
```

```
['1', '2', '3', ' ', 'PESU']
```

```
print(list(filter(lambda S:S>'',['1','2','3','',' ','','PESU'])))
```

```
['1', '2', '3', ' ', 'PESU']
```

```
def LessThan(S):
    return S<0
print(list(filter(LessThan,[1,-4,5,3,2])))
print(list(filter(LessThan,(-1,-4,-5,-3,2))))
```

```
[-4]
[-1, -4, -5, -3]
```

```
print(list(filter(lambda S:S<0,[1,-4,5,3,2])))
print(list(filter(lambda S:S<0,(-1,-4,-5,-3,2))))
```

```
[-4]
[-1, -4, -5, -3]
```

Selecting Elements From an Iterable With filter() in python

```
def Divisibleby3(X):
    return X % 3 ==0

print(list(filter(Divisibleby3,[2,4,6,8,10,12,3,17,19,144,256])))
```

```
[6, 12, 3, 144]
```

```
def ValidIdentifiers(S):
    return S.isidentifier()

List = ['One','Two','__PESU__','_PESU','1EC','EC1','True','False','while','not','1not']

print(list(filter(ValidIdentifiers,List)))
```

```
['One', 'Two', '__PESU__', '_PESU', 'EC1', 'True', 'False', 'while', 'not']
```

```
def ValidIdentifiers(S):
    return True

List = ['One','Two','__PESU__','_PESU','1EC','EC1']

print(list(filter(ValidIdentifiers,List)))
```

```
['One', 'Two', '__PESU__', '_PESU', '1EC', 'EC1']
```

```
def ValidIdentifiers(S):
    return False

List = ['One','Two','__PESU__','_PESU','1EC','EC1']

print(list(filter(ValidIdentifiers,List)))
```

```
[]
```

Reducing an Iterable to a Single Value With reduce() in python

- **reduce()** applies a function to the items in an iterable two at a time, progressively combining them to produce a single result.
- **Guido Van Rossum** actually advocated for **eliminating** all three of **reduce(), map()**, and **filter()** from python.
- **reduce()** is **no** longer a **built-in** function, but it's **available** for **import** from a **standard library module**
- To use **reduce()**, you need to import it from a module named **functools**.
- This is possible in **several** ways, but the following is the most straightforward
  - **from functools import reduce**
- The interpreter **places reduce()** into the **global namespace** and makes it available for use.

```
from functools import reduce

def Addition(X,Y):
    return(X + Y)

print('Simple Summation using reduce')
print(reduce(Addition,[0]))
print(reduce(Addition,[0,1]))
print(reduce(Addition,[0,1,2]))
print(reduce(Addition,[0,1,2,3,4]))

print('\nSimple Summation using sum')
print(sum([0]))
print(sum([0,1]))
print(sum([0,1,2]))
print(sum([0,1,2,3,4]))
```

```
Simple Summation using reduce
0
1
3
10

Simple Summation using sum
0
1
3
10
```

Reducing an Iterable to a Single Value With reduce() in python

- **reduce()** applies a function to the items in an iterable two at a time, progressively combining them to produce a single result.
- **Guido Van Rossum** actually advocated for **eliminating** all three of **reduce(), map()**, and **filter()** from python.
- **reduce()** is **no** longer a **built-in** function, but it's **available** for **import** from a **standard library module**
- To use **reduce()**, you need to import it from a module named **functools**.
- This is possible in **several** ways, but the following is the most straightforward
  - **from functools import reduce**
- The interpreter **places reduce()** into the **global namespace** and makes it available for use.

```
from functools import reduce

def Addition(X,Y):
    return(X + Y)

print('Concatenation using reduce')
print(reduce(Addition,['PESU']))
print(reduce(Addition,['PESU','P']))
print(reduce(Addition,['PESU','P','Section']))
print(reduce(Addition,['PESU','P','Secion','First','Semester']))

print('\nConcatenation using join')
print("".join(['PESU']))
print("".join(['PESU','P']))
print("".join(['PESU','P','Section']))
print("".join(['PESU','P','Secion','First','Semester']))
```

```
Concatenation using reduce
PESU
PESUP
PESUPSection
PESUPSecionFirstSemester

Concatenation using join
PESU
PESUP
PESUPSection
PESUPSecionFirstSemester
```

Reducing an Iterable to a Single Value With reduce() in python

- **reduce()** applies a function to the items in an iterable two at a time, progressively combining them to produce a single result.
- **Guido Van Rossum** actually advocated for **eliminating** all three of **reduce(), map()**, and **filter()** from python.
- **reduce()** is **no** longer a **built-in** function, but it's **available** for **import** from a **standard library module**
- To use **reduce()**, you need to import it from a module named **functools**.
- This is possible in **several** ways, but the following is the most straightforward
  - **from functools import reduce**
- The interpreter **places reduce()** into the **global namespace** and makes it available for use.

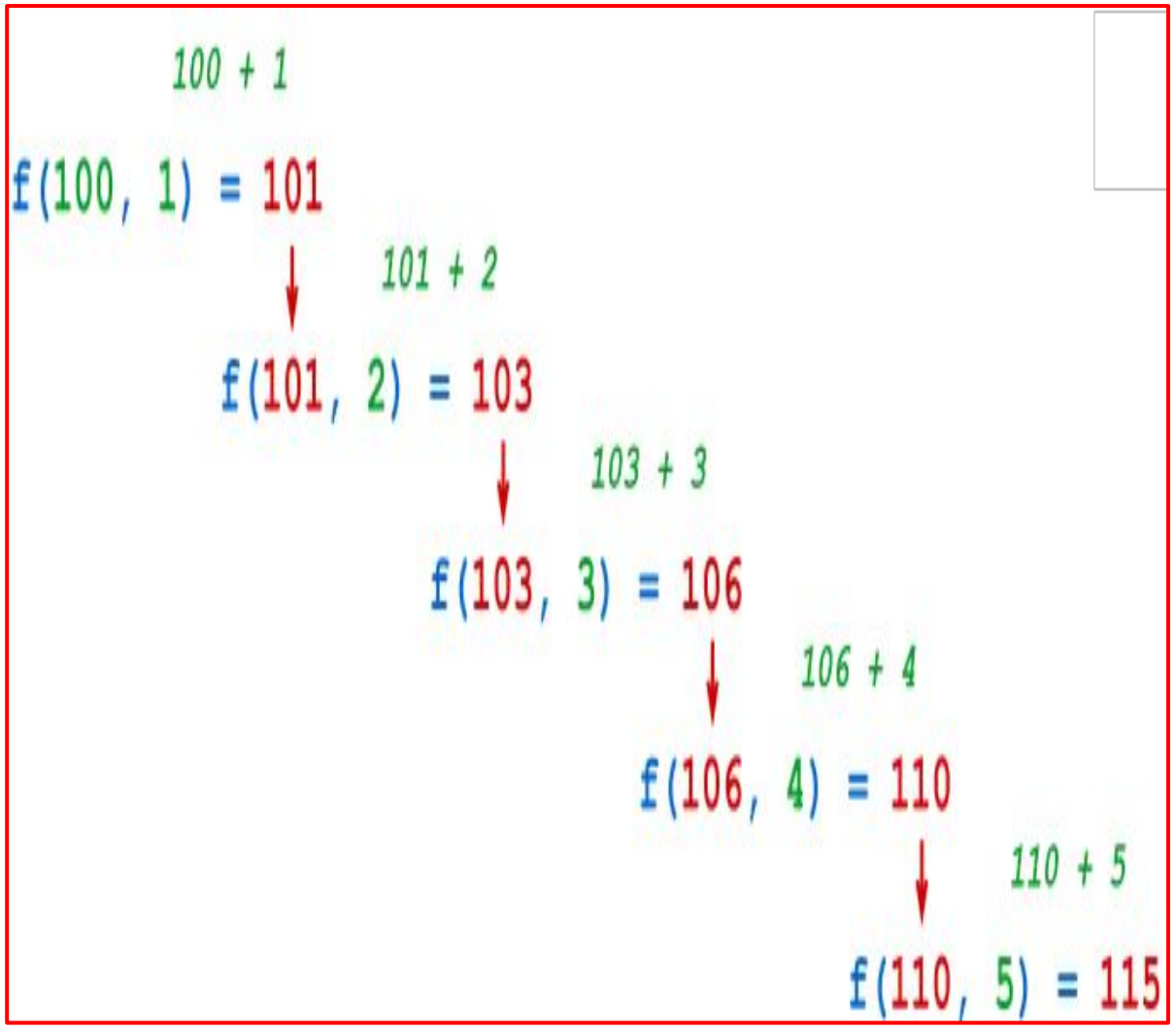
```
def Product(X,Y):
    return(X * Y)

def Factorial(n):
    from functools import reduce
    if(n==0) or (n==1):
        return 1
    return reduce(Product,range(1,n+1))

print('Implementation of Factorial using reduce')
print(Factorial(0))
print(Factorial(1))
print(Factorial(2))
print(Factorial(3))
print(Factorial(4))
print(Factorial(5))
```

```
Implementation of Factorial using reduce
1
1
2
6
24
120
```

Reducing an Iterable to a Single Value With reduce() in python




```
from functools import reduce

def Addition(X,Y):
    return(X + Y)

print(reduce(Addition,[],100))
print(reduce(Addition,[1],100))
print(reduce(Addition,[1,2],100))
print(reduce(Addition,[1,2,3,4,5],100))
print(reduce(Addition,[1,2,3,4,5],-10))
```

```
100
101
103
115
5
```

min() Function in python

- The python **min()** function returns the lowest value in a list of items.
- The **min()** function returns the **smallest item** in an **iterable**.
- **min()** function can also be used to find the smallest item between two or more parameters.
- The **min()** function has **two** forms:

  # to find the smallest item in an iterable
  - min(iterable, *iterables, key, default)

  # to find the smallest item between two or more objects
  - min(arg1, arg2, *args, key)

- **min() with iterable arguments -** min(**iterable**, ***iterables**, **key**, **default**)
  - **min() Parameters**
    - **iterable** - an iterable such as list, tuple, set, dictionary, etc.
    - ***iterables (optional)** - any number of iterables; can be more than one
    - **key (optional)** - key function where the iterables are passed and comparison is performed based on its return value
    - **default (optional)** - default value if the given iterable is empty

min() Function in python

- The python **min()** function returns the lowest value in a list of items.
- If we **pass** an **empty iterator**, a **ValueError** exception is raised.
- To avoid this, we can pass the **default** parameter.
- If we **pass more** than one **iterators**, the **smallest** item from the **given** iterators is **returned**.

```
MyList = [1,4,-5,3,2]
print('Minimum Value-->',min(MyList))
```

```
Minimum Value--> -5
```

```
Strings = ['One','Two','__PESU__','_PESU','1EC','EC1','True','False','while','not','1not']
print('Minimum Valued String-->',min(Strings))
```

```
Minimum Valued String--> 1EC
```

```
square = {2: 4, 3: 9, -1: 1, -2: 4}

# the smallest key
key1 = min(square)

print("The smallest key:", key1)

# the key whose value is the smallest
key2 = min(square, key = lambda k: square[k])


print("The key with the smallest value:", key2)

# getting the smallest value
print("The smallest value:", square[key2])
```

```
The smallest key: -2
The key with the smallest value: -1
The smallest value: 1
```

```
min('',default='Iterable is empty')
```

```
'Iterable is empty'
```

max() Function in python

- The python **max()** function returns the largest item in an iterable.
- It can also be used to find the largest item between two or more parameters
- The max() function has two forms:

  # to find the largest item in an iterable
  - max(iterable, *iterables, key, default)

  # to find the largest item between two or more objects
  - max(arg1, arg2, *args, key)

- **max() with iterable arguments -** max(**iterable**, ***iterables**, **key**, **default**)
  - **max() Parameters**
    - **iterable** - an iterable such as list, tuple, set, dictionary, etc.
    - ***iterables (optional)** - any number of iterables; can be more than one
    - **key (optional)** - key function where the iterables are passed and comparison is performed based on its return value
    - **default (optional)** - default value if the given iterable is empty

max() Function in python

- The python **max()** function returns the highest value in a list of items.
- If we **pass** an **empty iterator**, a **ValueError** exception is raised.
- To avoid this, we can pass the **default** parameter.
- If we **pass more** than one **iterators**, the **largest** item from the **given** iterators is **returned**.

```
MyList = [1,4,-5,3,2]
print('Maximum Value-->',max(MyList))
```

```
Maximum Value--> 4
```

```
Strings = ['One','Two','__PESU__','_PESU','1EC','EC1','True','False','while','not','1not']
print('Maximum Valued String-->',max(Strings))
```

```
Maximum Valued String--> while
```

```
square = {2: 4, 3: 9, -1: 1, -2: 4}

# the largest key
key1 = max(square)

print("The largest key:", key1)

# the key whose value is the largest
key2 = max(square, key = lambda k: square[k])


print("The key with the largest value:", key2)

# getting the largest value
print("The largest value:", square[key2])
```

```
The largest key: 3
The key with the largest value: 3
The largest value: 9
```

```
max('',default='Iterable is empty')
```

```
'Iterable is empty'
```

End of class #39, #40

Thank you

Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University
nitin.pujari@pes.edu

For Course Digital Deliverables visit www.pesuacademy.com