python for Computational Problem Solving - pCPS - Functional_Programming_Testing Lecture Slides - Class #43_#44

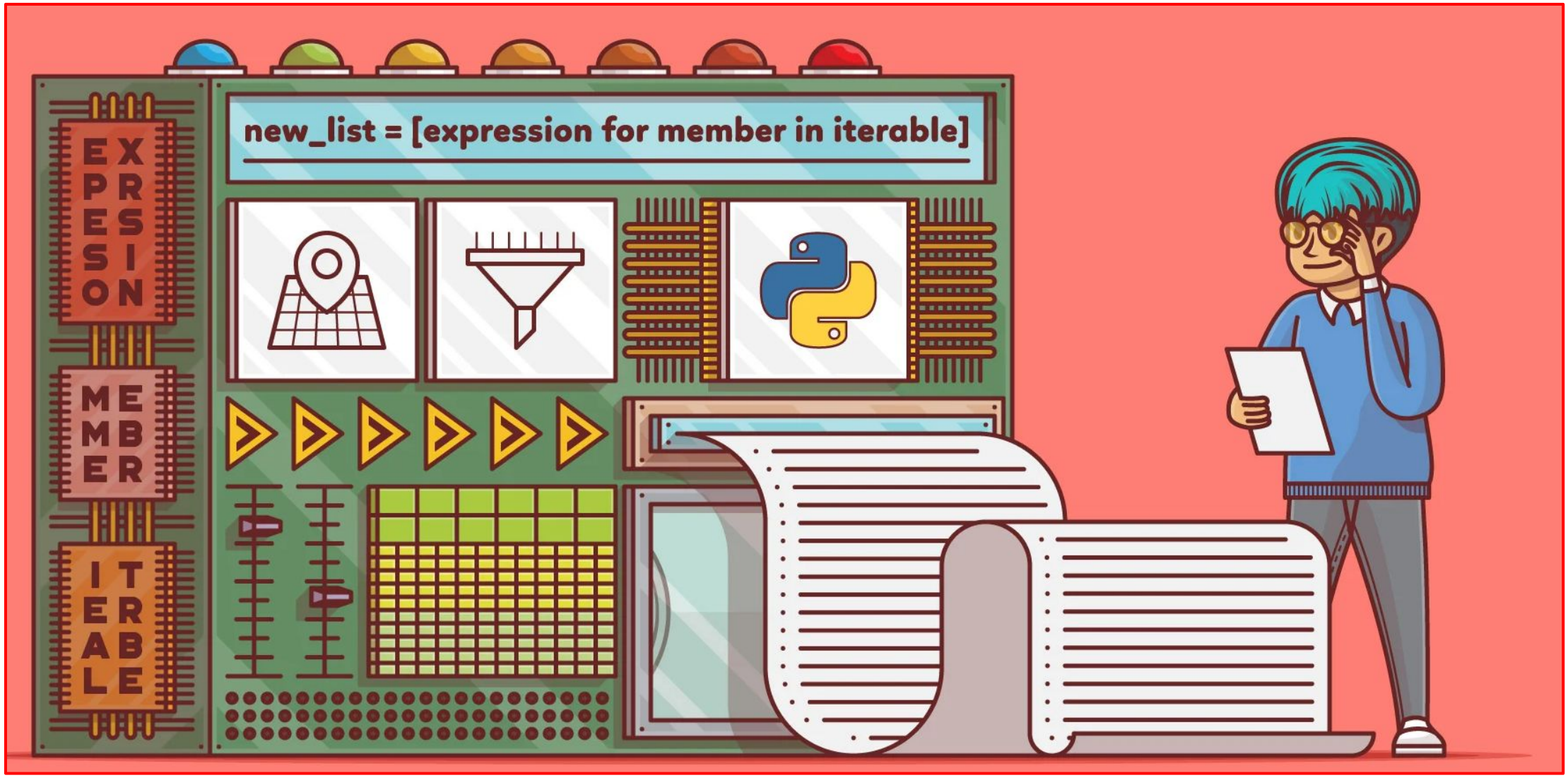
Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University

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
- Modules - import mechanisms
- list comprehension
- Generators and iterators
- Testing
  - Pytest , Function testing with Doctest
  - pdb debugger commands.

list Comprehension in python



list Comprehension in python

- One of the language's most distinctive features is the **list comprehension,** which you can use to create powerful functionality within a single line of code.

- To create a list, the most common type of loop is the **for** loop.
- You can use a for loop to create a list of elements in three steps:
  - **Instantiate** an empty list.
  - **Loop** over an iterable or range of elements.
  - **Append** each element to the end of the list.

- **map()** provides an alternative approach that's based in functional programming.

- You **pass** in a **function** and an **iterable**, and **map()** will **create** an **object**.

- This **object** contains the **output** you would get from running each iterable element through the supplied function.

```
Cubes=list()
for i in range(20):
    Cubes.append(i**3)
print(Cubes)
```

```
[0, 1, 8, 27, 64, 125, 216, 343, 512, 729, 1000, 1331, 1728, 2197, 2744, 3375, 4096, 4913, 5832, 6859]
```

```
Stocks_Transacted = [1004,56.45,1912,14.23,74123]
BrokerageFeesPercentage = 0.003

def Transaction(Volume):
    return(BrokerageFeesPercentage*Volume)

BrokerageFees = map(Transaction,Stocks_Transacted)
print(list(BrokerageFees))
```

```
[3.012, 0.16935, 5.736, 0.04269, 222.369]
```

list Comprehension in python

- **list comprehension** are a third way of creating lists.
- With this **elegant** approach, you could **rewrite** the for loop from the first example in just a single line of code
- The **syntax** for carrying our **list comprehension**
  - new_list = [**expression** for **member** in **iterable**]
- Every list comprehension in python includes three elements:
  - **expression** is the **member itself**, a **call** to a **method**, or any other **valid** expression that **returns** a **value**.
  - **member** is the **object** or **value** in the **list** or **iterable**.
  - **iterable** is a **list**, **set**, **sequence**, **generator**, or **any** other **object** that can **return** its **elements one** at a **time**.

```
Cubes = [i**3 for i in range(20)]
print(Cubes)
```

```
[0, 1, 8, 27, 64, 125, 216, 343, 512, 729, 1000, 1331, 1728, 2197, 2744, 3375, 4096, 4913, 5832, 6859]
```

```
Stocks_Transacted = [1004,56.45,1912,14.23,74123]
BrokerageFeesPercentage = 0.003

def Transaction(Volume):
    return(BrokerageFeesPercentage*Volume)

BrokerageFees = [Transaction(Element) for Element in Stocks_Transacted]
print(BrokerageFees)
```

```
[3.012, 0.16935, 5.736, 0.04269, 222.369]
```

list Comprehension in python

- The only distinction between this implementation and **map()** is that the **list comprehension** returns a **list**, **not** a **map object**.

```
Stocks_Transacted = [1004,56.45,1912,14.23,74123]
BrokerageFeesPercentage = 0.003

def Transaction(Volume):
    return(BrokerageFeesPercentage*Volume)

BrokerageFees = map(Transaction,Stocks_Transacted)
print('Returns-->',type(BrokerageFees))
print(list(BrokerageFees))
```

```
Returns--> <class 'map'>
[3.012, 0.16935, 5.736, 0.04269, 222.369]
```

```
Stocks_Transacted = [1004,56.45,1912,14.23,74123]
BrokerageFeesPercentage = 0.003

def Transaction(Volume):
    return(BrokerageFeesPercentage*Volume)

BrokerageFees = [Transaction(Element) for Element in Stocks_Transacted]
print('Returns-->',type(BrokerageFees))
print(BrokerageFees)
```

```
Returns--> <class 'list'>
[3.012, 0.16935, 5.736, 0.04269, 222.369]
```

list Comprehension in python - Benefits

- **List comprehensions** are often **described** as being more **pythonic** than **loops** or **map()**
- python **Idioms** is for people coming from other languages and how to improve your **idiomatic** practices with python.
- **Idioms**: a group of words established by usage as having a meaning not deducible from those of the individual words
- **Idiomatic** :using, containing, or denoting expressions that are natural to a native speaker
- **Benefit #1**:
  - One main benefit of using a list comprehension is that, it's a **single** tool that you can use in many different situations.
  - In addition to standard list creation, **list comprehension**s can also be used for **mapping** and **filtering**. You don't have to use a different approach for each scenario.
- **Benefit #2**:
  - **List comprehensions** are also more **declarative** than loops, which means they're **easier** to **read** and **understand**.
    - Loops require you to focus on how the list is created. You have to manually create an empty list, loop over the elements, and add each of them to the end of the list.
  - With a list comprehension, you can instead **focus** on what **you want** to go in the list and **trust** that **python** will take care of how the list construction takes place.

list Comprehension in python - Conditionals

- A more complete description of the comprehension formula adds support for optional conditionals.

- The most common way to add **conditional** logic to a list comprehension is to add a conditional to the end of the expression:
  new_list = [expression for member in iterable (if conditional)]

- **Conditionals** are important because they allow list comprehensions to **filter** out **unwanted** values, which would normally require a call to **filter()**

```
import sys
print('python version installed-->',sys.version)
```

```
python version installed--> 3.8.12 (default, Oct 12 2021, 13:49:34)
[GCC 7.5.0]
```

```
lower = list('abcdefghijklmnopqrstuvwxyz')
UPPER = list(('abcdefghijklmnopqrstuvwxyz'.upper()))
Alphabets = lower + UPPER

AlphabetSet = set(Alphabets)
VowelSet = set(list('aeiou')+ list('AEIOU'))
ConsonantSet = AlphabetSet - VowelSet

Sentence = 'PES University First Semester P Section'

print('Given Sentence-->',Sentence)

ExtractVowels = [Symbol for Symbol in Sentence if (Symbol in VowelSet)]
print('Vowel Sentence-->',''.join(ExtractVowels))

ExtractConsonants = [Symbol for Symbol in Sentence if (Symbol in ConsonantSet)]
print('Consonant Sentence-->',''.join(ExtractConsonants))
```

```
Given Sentence--> PES University First Semester P Section
Vowel Sentence--> EUieiieeeeio
Consonant Sentence--> PSnvrstyFrstSmstrPSctn
```

list Comprehension in python - Conditionals

- A more complete description of the comprehension formula adds support for optional conditionals.

- The most common way to add **conditional** logic to a list comprehension is to add a conditional to the end of the expression:
  new_list = [expression for member in iterable (if conditional)]

- **Conditionals** are important because they allow list comprehensions to **filter** out **unwanted** values, which would normally require a call to **filter()**

```
lower = list('abcdefghijklmnopqrstuvwxyz')
UPPER = list(('abcdefghijklmnopqrstuvwxyz'.upper()))
Alphabets = lower + UPPER
VowelSet = set(list('aeiou')+ list('AEIOU'))

Sentence = 'PES University First Semester P Section'

def Consonant(Symbol):
    return(Symbol in Alphabets)and (Symbol not in VowelSet)

ExtractConsonants = [Symbol for Symbol in Sentence if Consonant(Symbol)]

ExtractVowels = [Symbol for Symbol in Sentence if not Consonant(Symbol)]
ExtractVowels = [Symbol for Symbol in ExtractVowels if not Symbol.isspace()]

print('Given Sentence-->',Sentence)
print('Vowel Sentence-->','.join(ExtractVowels))
print('Consonant Sentence-->','.join(ExtractConsonants))
```

```
Given Sentence--> PES University First Semester P Section
Vowel Sentence--> EUieiieeeeio
Consonant Sentence--> PSnvrstyFrstSmstrPSctn
```

list Comprehension in python - Conditionals

- You can place the conditional at the end of the statement for simple filtering

- It is also useful to place the conditional near the beginning of the expression

- With this formula, you can use conditional logic to select from multiple possible output options.:

new_list = [expression (if conditional) for member in iterable]

```
Stocks_Transacted = [1004,56.45,1912,14.23,74123]
BrokerageFeesPercentage = 0.003

Stocks_Considered = [Element if(Element>100) else 'NA' for Element in Stocks_Transacted]

print('Returns-->',type(Stocks_Considered))
print(Stocks_Considered)
```

```
Returns--> <class 'list'>
[1004, 'NA', 1912, 'NA', 74123]
```

```
Stocks_Transacted = [1004,56.45,1912,14.23,74123]
BrokerageFeesPercentage = 0.003

def Stocks(Element):
    return Element if Element>100 else 'NA'

Stocks_Considered = [Stocks(Element) for Element in Stocks_Transacted]

print('Returns-->',type(Stocks_Considered))
print(Stocks_Considered)
```

```
Returns--> <class 'list'>
[1004, 'NA', 1912, 'NA', 74123]
```

PES UNIVERSITY ONLINE

PES UNIVERSITY ONLINE

End of class #43, #44

Thank you

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

For Course Digital Deliverables visit www.pesuacademy.com