Department of Computer Science and Engineering
PES University, Bangalore, India

Python For Computational Problem Solving (UE22CS151A)

Mr. Prakash C O
Asst. Professor,
Dept. of CSE, PESU
coprakasha@pes.edu

Comprehensions in Python

Comprehensions in Python provide us with a short and concise way to construct new sequences (such as lists, set, dictionary etc.) using sequences which have been already defined.

Python supports the following 4 types of comprehensions:

```
* List Comprehensions
* Dictionary Comprehensions
* Set Comprehensions
* Generator Comprehensions
```

In set theory, we learn a couple of ways of creating a set.

```
1) Enumeration: All the elements are listed.
    Example: s = { 1, 3, 5, 7, 9}

2) Rule based or builder method:
    s = { x | x is odd and 1 <= x <= 10 }
```

How to Create Lists in Python

Problem statement: Create a list containing the first ten perfect squares.

Solution-01: By enumeration: All the elements are listed.

In [ ]:
```
squares = [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Solution-02: Using for Loops

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:

1. Instantiate an empty list.
2. Loop over an iterable or range of elements.
3. Append each element to the end of the list.

In [1]:
```
squares = []
for n in range(1,11):
    squares.append(n*i)

print(squares)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Solution-03: Using map() Objects

In [2]:
```
squares = list(map(lambda n : n*n, range(1,11)))
print(squares)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

List comprehension in Python

In this section, we shall learn how to create lists using rules. This method of list creation is called list comprehension.

List comprehensions provide an elegant way to create new lists from existing squences.

The following is the basic structure of a list comprehension:

new_list = [expression for member in iterable]

The above list comprehension includes three elements:

1. expression: expression is the member itself, or a call to a method, or any other valid expression that returns a value.
2. member: member is the object or value in the iterable.
3. iterable: iterable is a list, set, sequence, generator, or any other object that can return its elements one at a time.

Semantically, new_list = [expression for member in iterable] is equivalent to the following.

1. Create an empty list.
2. Execute for part of the list comprehension and for every iteration
   - Evaluate the expression, and append the expression result to the list.
3. The result is the new list so created.

Create a list containing the first ten perfect squares.

Solution-04: By List Comprehension

In [3]:
```
squares = [ n*n for n in range(1,11) ]
print(squares)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

List Comprehension Using Conditional Logic

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 at the end.

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

Semantically, new_list = [expression for member in iterable (if conditional)] is equivalent to the following.

1. Create an empty list.
2. Execute for part of the list comprehension and for every iteration
   - Evaluate the if_condition.
   - If if_condition is satisfied then evaluate the expression and append the expression result to the list.
3. The result is the new list so created.

Here, your conditional statement comes just before the closing bracket.

Conditionals are important because they allow list comprehensions to filter out unwanted values.

Note:

Python is famous for allowing you to write code that's elegant, easy to write, and almost as easy to read as plain English. 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.

Example 1:

Create an output list which contains only the even numbers which are present in the input list using list comprehension

Using List Comprehension:

In [6]:
```
input_list = [1, 2, 3, 4, 5, 6, 7, 9, 10]

output_list = [i for i in input_list if i % 2 == 0]
print(output_list)
```

```
[2, 4, 6, 10]
```

Using filter() function:

In [9]:
```
# Approach-01: Using filter()
input_list = [1, 2, 3, 4, 5, 6, 7, 9, 10]

def even(num):
    return num % 2 == 0

output_list = list(filter(even,input_list))
print(output_list)
```

```
[2, 4, 6, 10]
```

In [11]:
```
# Approach-02: Using filter() and lambda function
input_list = [1, 2, 3, 4, 5, 6, 7, 9, 10]

output_list = list(filter(lambda num : num % 2 == 0, input_list))
print(output_list)
```

```
[2, 4, 6, 10]
```

Example 2: Generate multiplication table for a given number.

In [5]:
```
n=5
table = [ n * i for i in range(1,11)]
print(table)
```

```
[5, 10, 15, 20, 25, 30, 35, 40, 45, 50]
```

Example 3: Compute list of tuples having a number and its square using list comprehension

In [1]:
```
n = int(input('Enter value for n:'))
l3 = [ (x, x * x) for x in range(1,n+1)]
print(l3)
```

```
Enter value for n:10
[(1, 1), (2, 4), (3, 9), (4, 16), (5, 25), (6, 36), (7, 49), (8, 64), (9, 81), (10, 10
0)]
```

Example 4: Compute the list of tuple of strings and its length using list comprehension

In [2]:
```
city_list = ['bangalore', 'mysore', 'hubballi', 'shivamogga']

cityname_length = [ (name, len(name)) for name in city_list]
print(cityname_length)
```

```
[('bangalore', 9), ('mysore', 6), ('hubballi', 8), ('shivamogga', 10)]
```

The one below is an example of nested loops in list comprehension.

Example 5: Compute cartesian product

The Cartesian product of two sets A and B, denoted A × B, is the set of all ordered pairs (a, b) where a is in A and b is in B. where A = {0,1,2,3} B = {0,1,2,3}

In [6]:
```
l5 = [ (a, b) for a in range(4) for b in range(4)]
print(l5)
```

```
[(0, 0), (0, 1), (0, 2), (0, 3), (1, 0), (1, 1), (1, 2), (1, 3), (2, 0), (2, 1), (2, 2),
(2, 3), (3, 0), (3, 1), (3, 2), (3, 3)]
```

The one below is an example of selection amongst the many produced by the loops. Observe the similarity with filter. In fact it is a combination of map and filter.

Example 6: Given A = {0,1,2,3} and B = {0,1,2,3}

Show the partial order relation a<b where a is in A and b is in B.

In [1]:
```
l6 = [ (a, b) for a in range(4) for b in range(4) if a < b]
print(l6)
```

```
[(0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 3)]
```

Example 7: Program to convert all names in list to uppercase using list comprehension

In [16]:
```
# map
city_list = ['bangalore', 'mysore', 'hubballi', 'shivamogga' ]

print(list(map(str.upper,city_list)))
```

```
['BANGALORE', 'MYSORE', 'HUBBALLI', 'SHIVAMOGGA']
```

In [2]:
```
# list comprehension
city_list = ['bangalore', 'mysore', 'hubballi', 'shivamogga' ]


b = [ name.upper() for name in city_list ]
print(b)
```

```
['BANGALORE', 'MYSORE', 'HUBBALLI', 'SHIVAMOGGA']
```

Example 8: find all words in list whose len exceeds 7

In [2]:
```
city_list = ['bangalore', 'mysore', 'hubballi', 'shivamogga' ]
```

In [3]:
```
b = [ name for name in city_list if len(name) > 7]
print(b)
```

```
['bangalore', 'hubballi', 'shivamogga']
```

Example 9: convert all words in list to uppercase if len exceeds 7

In [6]:
```
city_list = ['bangalore', 'mysore', 'hubballi', 'shivamogga' ]
b = [ name.upper() for name in city_list if len(name) > 7]
print(b)
```

```
['BANGALORE', 'HUBBALLI', 'SHIVAMOGGA']
```

Example 10: Generate a nested list [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

In [20]:
```
b = [ [x, x+1, x+2] for x in range(1,10,3)]
print(b)
```

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Example 11: Find common numbers from 2 lists

In [19]:
```
l1=[1,2,3,4]
l2=[3,4,5,6]

l3=[ x for x in l1 if x in l2]
print(l3)
```

```
[3, 4]
```

In [20]:
```
l1=[1,2,3,4]
l2=[3,4,5,6]
l4=[ x for x in l1 for y in l2 if x==y]
print(l3)
```

```
[3, 4]
```

Example 12: filter all characters in sentence that aren't a vowel

In [3]:
```
sentence = 'The rocket, who was named Ted, came back \
            from Mars because he missed his friends.'


vowels = [ch for ch in sentence if ch in 'aeiouAEIOU']
print(vowels)
```

```
['e', 'o', 'e', 'o', 'a', 'a', 'e', 'e', 'a', 'e', 'a', 'o', 'a', 'e', 'a', 'u', 'e',
'e', 'i', 'e', 'i', 'i', 'e']
```

Example 12a: filter all vowels and whitespace in sentence.

In [5]:
```
# Approach-01
sentence = 'The rocket, who was named Ted, came back \
            from Mars because he missed his friends.'
vowels = [ch for ch in sentence if ch not in ' aeiouAEIOU']
print(vowels)
```

```
['T', 'h', 'r', 'c', 'k', 't', ',', 'w', 'h', 'w', 's', 'n', 'm', 'd', 'T', 'd', ',',
'c', 'm', 'b', 'c', 'k', 'f', 'r', 'm', 'M', 'r', 's', 'b', 'c', 's', 'h', 'm', 's',
's', 'd', 'h', 's', 'f', 'r', 'n', 'd', 's', '.']
```

In [15]:
```
# Approach-02
```

```
sentence = 'The rocket, who was named Ted, came back \
            from Mars because he missed his friends.'
def is_consonant(letter):
    vowels = 'aeiouAEIOU'
    return letter.isalpha() and letter not in vowels

consonants = [i for i in sentence if is_consonant(i)]
print(consonants)
```

```
['T', 'h', 'r', 'c', 'k', 't', 'w', 'h', 'w', 's', 'n', 'm', 'd', 'T', 'd', 'c', 'm',
'b', 'c', 'k', 'f', 'r', 'm', 'M', 'r', 's', 'b', 'c', 's', 'h', 'm', 's', 's', 'd',
'h', 's', 'f', 'r', 'n', 'd', 's']
```

You can place the conditional at the end of the statement for simple filtering, but what if you want to change a member value instead of filtering it out? In this case, it's useful to place the conditional near the beginning of the expression:

new_list = [on_true_expr if condition else on_false_expr for member in iterable]

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

Example:

If you have a list of prices, then you may want to replace negative prices with 0 and leave the positive values unchanged:

In [18]:
```
original_prices = [1.25, -9.45, 10.22, 3.78, -5.92, 1.16]

new_prices = [price if price > 0 else 0 for price in original_prices]
print(new_prices)
```

```
[1.25, 0, 10.22, 3.78, 0, 1.16]
```

In [18]:
```
def get_price(price):
    return price if price > 0 else 0

prices = [get_price(i) for i in original_prices]
print(prices)
```

```
[1.25, 0, 10.22, 3.78, 0, 1.16]
```

Using the Walrus Operator

Python 3.8 will introduce the assignment expression, also known as the walrus operator. To understand how you can use it, consider the following example.

Say you need to make ten requests to an API that will return temperature data. You only want to return results that are greater than 100 degrees Fahrenheit. Assume that each request will return different data. In this case, there's no way to use a list comprehension in Python to solve the problem.

The formula

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

provides no way for the conditional to assign data to a variable that the expression can access. The walrus operator solves this problem.

The walrus operator(:=) is an operator used in assignment expressions, which can return the value being assigned, unlike traditional assignment statements.

Walrus operator allows you to run an expression while simultaneously assigning the output value to a variable.

In [24]:
```
(test2 := False)
```

Out[24]:
```
False
```

In [1]:
```
# Generate a list having 20 temperatures that are between 90-110 degrees Fahrenheit
import random
def get_weather_data():
    return random.randrange(90, 110)

temps = [ get_weather_data() for _ in range(20) ]
print(temps)
```

```
[106, 95, 98, 90, 103, 92, 91, 99, 100, 93, 104, 95, 105, 107, 99, 95, 90, 91, 109, 108]
```

In [25]:
```
# Generate a list having 20 temperatures that are between 90-110 degrees Fahrenheit
import random
def get_weather_data():
    return random.randrange(90, 110)

temps = [(temp := get_weather_data()) for _ in range(20) ]
print(temps)
```

```
[90, 94, 100, 105, 94, 104, 92, 92, 92, 107, 102, 97, 103, 96, 92, 90, 94, 105, 99, 97]
```

The following example shows how this is possible, using get_weather_data() to generate fake weather data:

In [2]:
```
import random
def get_weather_data():
    return random.randrange(90, 110)

hot_temps = [temp for _ in range(10) if (temp := get_weather_data()) >= 100]
print(hot_temps)
```

```
[108, 105, 104, 104]
```

You won't often need to use the assignment expression inside of a list comprehension in Python, but it's a useful tool to have at your disposal when necessary.

List comprehension provides an alternate mechanism to functional programming constructs map and filter.

We also have set, dict comprehension. These are not part of the course – hence not discussed.

Benefits of Using List Comprehensions

- One main benefit of using a list comprehension in Python is that it's a single tool that you can use in many different situations.
- In addition to standard list creation, list comprehensions can also be used for mapping and filtering. You don't have to use a different approach for each scenario.

This is the main reason why list comprehensions are considered Pythonic, as Python embraces simple, powerful tools that you can use in a wide variety of situations. As an added side benefit, whenever you use a list comprehension in Python, you won't need to remember the proper order of arguments like you would when you call map().

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 in Python,

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.

Exercises

1)Form a list of two letter shortname made of the first and last characters from a given list of names, if the length of the name is greater than or equal to 3.

In [3]:
```
names_list = ['Ronaldo','Messi','Naymer','Mbappe']
shortnames_list = [ name[0]+name[-1] for name in names_list if len(name)>=3]
print(shortnames_list)
```

```
['Ro', 'Mi', 'Nr', 'Me']
```

2)Count the number of spaces in a given multiword string.

In [5]:
```
string="Regarded as one of the best footballers of the current generation"
space_count=len([ch for ch in string if ch==' '])
print(space_count)
```

```
10
```

3)Find all numbers which are both odd and palindromes between a pair of numbers 20 and 100 (both inclusive).

In [7]:
```
oddpaln_list=[num for num in range(20,101) if num%2!=0 and str(num)==str(num)[::-1]]
print("The numbers which are both odd and palindromes are: ",oddpaln_list)
```

```
The numbers which are both odd and palindromes are:  [33, 55, 77, 99]
```

4)Create a new list of strings made of the first 2 and last 2 characters From a given list of words only if the length of the string is greater than 4.

In [10]:
```
fruits = ['pear','apple','banana','kiwi','orange','pappaya']

new_fruits = [ f[0:2]+f[-2:] for f in fruits if len(f)>4]
print(new_fruits)
```

```
['aple', 'bana', 'orge', 'paya']
```

5)Find and display the intersection of two lists.

In [11]:
```
l1 = [1,2,3,4,5]
l2 = [4,5,6,7,8]
l3 = [x for x in l1 if x in l2]
print(l3)
```

```
[4, 5]
```

6)

In [15]:
```
l=['Messi','Mbappe','Benzema','Bruyne']
#print(max(l,key=lambda s:len(s)))
print(max(l,key=len))
print(min(l,key=len))
```

```
Benzema
Messi
```

In [12]:
```
l1=[1,2,3,4]
l2=['one','two','three','four']
```

```
print(list(zip(l1,l2)))
```

```
[(1, 'one'), (2, 'two'), (3, 'three'), (4, 'four')]
```

In [14]:
```
l1=[1, 2, 3, 4]
l2=['one','two','three','four']
l3 = zip(l1,l2)
l4, l5 = zip(*l3) # unpacking operation
print(l4, l5)
```

```
(1, 2, 3, 4) ('one', 'two', 'three', 'four')
```

Set comprehension

- A set comprehension is almost exactly the same as a list comprehension in Python.
- The difference is that set comprehensions make sure the output contains no duplicates.
- You can create a set comprehension by using curly braces instead of brackets:

In [21]:
```
quote = "life, uh, finds a way"
unique_vowels = {ch for ch in quote if ch in 'aeiou'}
print(unique_vowels)
```

```
{'i', 'u', 'a', 'e'}
```

Your set comprehension outputs all the unique vowels it found in quote. Unlike lists, sets don't guarantee that items will be saved in any particular order.

Dict comprehension

- Dictionary comprehension is similar, with the additional requirement of defining a key.
- Dictionary comprehension is defined with a key:value pair in expression.

The following is the basic structure of a dictionary comprehension:

new_dict = { expr1 : expr2 for member(s) in iterable (if conditional) }

Example 1:

- states_list = ['Karnataka','Gujarat', 'Maharashtra', 'Rajasthan']
- capitals_list = ['Bangalore','Gandhinagar', 'Mumbai', 'Jaipur']

Using above lists, create a new dictionary having state name as key and state capital as value using dictionary comprehension.

In [5]:
```
states_list = ['Karnataka','Gujarat', 'Maharashtra', 'Rajasthan']
capitals_list = ['Bangalore','Gandhinagar', 'Mumbai', 'Jaipur']

dict1 = {state_name:capital_name for state_name,capital_name in zip(states_list, capital
print(dict1)
```

```
{'Karnataka': 'Bangalore', 'Gujarat': 'Gandhinagar', 'Maharashtra': 'Mumbai', 'Rajastha
n': 'Jaipur'}
```

In [1]:
```
states_list = ['Karnataka','Gujarat', 'Maharashtra', 'Rajasthan']
capitals_list = ['Bangalore','Gandhinagar', 'Mumbai', 'Jaipur']

dict2 = {states_list[i]:capitals_list[i] for i in range(len(states_list))}
print(dict2)
```

```
{'Karnataka': 'Bangalore', 'Gujarat': 'Gandhinagar', 'Maharashtra': 'Mumbai', 'Rajastha
```

```
n': 'Jaipur'}
```

Example 2:

Create a new dictionary which contains only the odd numbers that are present in the input list as keys and their cubes as values.

In [16]:
```
input_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

dict2 = {num : num ** 3 for num in input_list if num % 2 != 0}
print(dict2)
```

```
{1: 1, 3: 27, 5: 125, 7: 343, 9: 729}
```

Comprehensions can be nested to create combinations of lists, dictionaries, and sets within a collection.

For example, say a climate laboratory is tracking the high temperature(in fahrenheit) in five different cities for the first week of June.

In [2]:
```
cities = ['Delhi', 'Bangalore', 'Chennai', 'Mumbai', 'Hyderabad']
import random

temps = {city: [random.randrange(90, 110) for _ in range(7)] for city in cities}
print(temps)
```

```
{'Delhi': [107, 96, 108, 97, 103, 106, 105], 'Bangalore': [101, 92, 96, 96, 104, 92, 9
3], 'Chennai': [96, 108, 94, 106, 108, 104, 92], 'Mumbai': [107, 101, 96, 105, 100, 94,
104], 'Hyderabad': [107, 102, 104, 109, 106, 92, 100]}
```

References: